Escola Politècnica Superior
d'Edificació de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Enginyeria geomàtica i topografia

Treball de fi de grau

Epipolarització d'un parell fotogramètric fent ús de la matriu fonamental

Projectista: Joel Egea Quijada

Director: Albert Prades Valls

Convocatòria: Octubre 2017

Resum

Aquest treball de fi de grau consisteix en la creació d'un programa escrit en C++ fent ús de les llibreries OpenCV que ens permeti trobar la geometria epipolar d'un parell fotogramètric sense conèixer els paràmetres d'orientació de la càmera amb la que s'han realitzat les fotografies. Aquest programa permetrà realitzar aquest procés de manera automàtica introduint dues imatges.

Primerament es realitza una detecció de punts d'interès en les dues imatges i després es troben les parelles de punts homòlogues, les quals posteriorment són refinades eliminant les que realment no són bones. Finalment, amb l'ajut de la matriu fonamental, obtenim la geometria epipolar de les imatges.

Abstract

This end-of-degree project consist in coding a program in C++ using the OpenCV libraries that allow us to find the epipolar geometry of a photogrammetric pair without knowing the orientation's parameters of the camera with which the photos were taken. This program will allow us to perform this process automatically by entering two images.

Firstly, there is a detection of keypoints in the two images and then the pairs of homologous points are found, which are subsequently refined, eliminating those that are not really good. Finally, with the help of the fundamental matrix, we obtain the epipolar geometry of the images.

Índex

Introducció

L'objectiu principal del projecte és, a partir un programa escrit en C++ i ajudat amb OpenCV, trobar la geometria epipolar de les dues imatges introduïdes al programa sense saber els paràmetres d'orientació interns de la càmera ni els paràmetres externs. Per poder realitzar aquest procés utilitzarem la matriu fonamental.

Això ens permet agafar un parell d'imatges d'internet, d'una base de dades o d'on sigui sense saber l'origen d'aquestes i poder orientar-les relativament.

Llavors en aquest projecte partiré d'aquestes dues imatges (Fig. 1 i Fig. 2). Com es pot observar és una carta de la baralla espanyola, concretament, la sota d'espases. La carta de la segona imatge està sotmesa a una rotació, translació i canvi d'escala. Les fotografies s'han realitzat amb el meu propi mòbil amb una càmera posterior de 13 Mpx de resolució i sensor Samsung S5K3L2.

Les imatges s'utilitzaran amb el programa que he escrit, seguint els passos convenients.

Tots els passos realitzats al programa són descrits a continuació i es van mostrant els resultats que es van obtenint al realitzar cada pas.

FIG. 1 PRIMERA IMATGE AMB L'OBJECTE

FIG. 2 SEGONA IMATGE AMB L'ESCENARI

Nucli de la memòria

Què és OpenCV?

OpenCV (*Open Source Computer Vision Library*)(Bradski 2000) és un conjunt de llibreries creades per Intel, destinades a la computació d'aplicacions en temps real. Compta amb més de 2500 algorismes optimitzats, que poden ser utilitzats com per exemple: detecció i reconeixement de cares humanes, identificació d'objectes, seguiment de moviments d'objectes, trobar imatges similars en una base de dades d'imatges, etc.

OpenCV és gratuït per a fins acadèmics i comercials, s'utilitza sota la llicencia BSD (*Berkeley Software Distribution*). Suporta les plataformes de Windows, Linux, Mac OS, iOS i Android. Va ser escrit, originalment, en C++, però es pot utilitzar en Python, Java, C, C++ i MATLAB.

Instal·lació OpenCV

Primerament descarreguem i instal·lem Microsoft Visual Studio, que concretament s'ha utilitzat per realitzar aquest projecte la versió Microsoft Visual Studio 2010. Aquest entorn de programació ens servirà per escriure tot el codi per implementar el programa.

A continuació, descarreguem OpenCV de la web oficial (www.opencv.org), amb el qual obtindrem una sèrie de llibreries que ens permetrà operar amb imatges.

En aquest cas s'ha utilitzat la versió 2.4.13, per la comoditat de que tots els mòduls ja venen inclosos en l'instal·lador, en canvi a la versió més recent (3.2.0) no tots els mòduls venen amb l'instal·lador, concretament hi ha unes llibreries que necessitem que són denominades *nonfree* que s'han d'instal·lar apart.

Una vegada descarregat, extraiem l'arxiu i el posem al directori que vulguem, preferiblement s'hauria de posar a l'arrel del disc dur, que sol ser: *"C:\"*, ja que és una ruta curta i ens ajudarà a evitar problemes.

Seguidament anem a les propietats del PC (Fig. 3 Propietats del PC i entrem dins de "*Configuració*

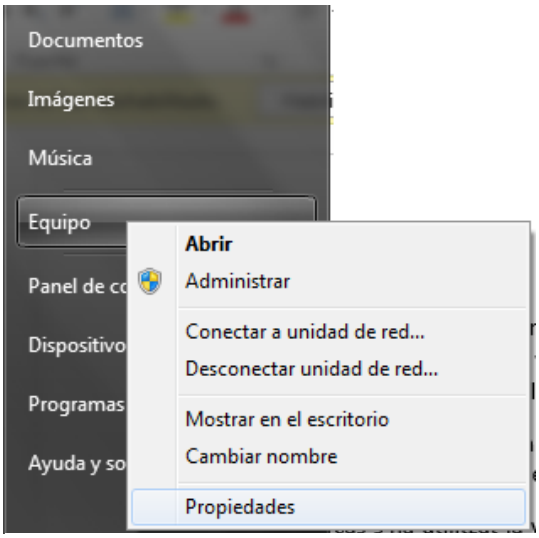

FIG. 3 PROPIETATS DEL PC

avançada del sistema" després a "*Variables de l'entorn*".

Aquí crearem la nova variables fent clic a "Nova...". Depenent de si el programa l'utilitzarem en nomes un usuari o en tot el sistema crearem la variable en una de les 2 opcions. I escrivim OPENCV_DIR com a nom de la variable i a sota la ruta del arxius binaris i llibreries de OpenCV, com hi apareix a la imatge de sota (Fig. 4).

Hem de tenir en compte l'arquitectura del nostre sistema operatiu a l'hora d'escriure la ruta. Escriurem x64 si el nostre sistema operatiu és de 64 bits i x86 si el sistema operatiu és de 32 bits.

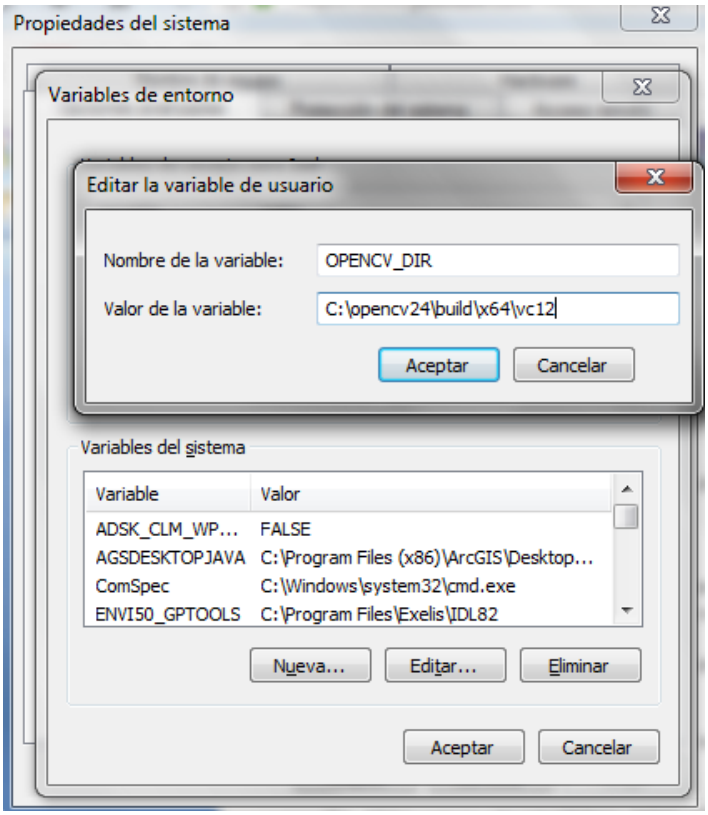

FIG. 4 VARIABLES DE L'ENTORN

A l'apartat de Variables del sistema hem d'editar la variable *PATH*, per afegir la ruta de la carpeta amb nom bin de OpenCV, que en el meu cas és aquesta: ";%OPENCV_DIR%\bin".

S'escriu ";" al davant per separar les rutes que contenen aquesta variable.

Ja tenim instal·lat el programa, ara hem de configurar el Microsoft Visual Studio, que en el meu cas és la versió de 2010 (Microsoft Visual Studio 2010).

Ara obrim el Microsoft Visual Studio i creem un nou projecte, concretament, una aplicació de consola de Win32. A continuació ens dirigim al menú de l'esquerra anomenat *Administrador de propietats* (Fig. 5).

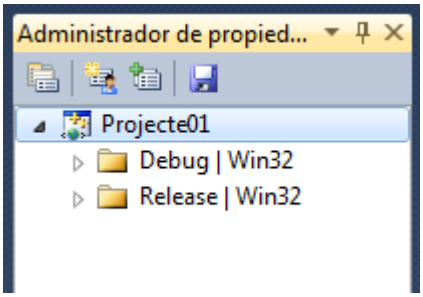

FIG. 5 ADMINISTRADOR DE PROPIETATS

Primerament crearem una plantilla per a les propietats del projecte, així cada cop que creem un nou projecte en el qual volem utilitzar OpenCV només haurem de carregar la nostra plantilla de propietats que contindrà totes les modificacions a les propietats fetes la primera vegada.

Llavors fem clic dret al nom del nostre projecte per accedir a *Afegir nova fulla de propietats de projecte...*, li donem un nom i la guardem. Com he dit abans aquesta plantilla guardarà tots el canvis que fem a les propietats.

La propera vegada que vulguem obrir un nou projecte per treballar amb l'OpenCV només haurem d'anar a la tercera opció de la barra de l'*Administrador de propietats* com s'indica a la imatge de sota (Fig. 6). I carregar la nostra plantilla, per no fer cada vegada que obrim un nou projecte el que ve a continuació.

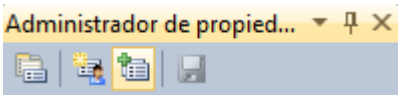

FIG. 6 BARRA D'ADMINISTRADOR DE PROPIETATS

Una vegada tenim preparada la nostra plantilla fem clic dret a *Debug | Win32* per obrir les propietats i modificarem algunes coses per poder utilitzar OpenCV dintre d'aquest programa. Obrim la pestanya de *C/C++* i anem a *General*, al primer apartat (*Directoris d'inclusió addicionals*) escriurem la ruta de la carpeta *Include* de OpenCV, que en el meu cas és aquesta "C:\opencv24\build\include".

Seguidament, anem a la pestanya *Vinculador* i entrem a *General*, a l'apartat de *Directoris de biblioteques addicionals* introduirem la ruta de la carpeta que conté les llibreries de l'OpenCV. En aquest cas, podem utilitzar la variable de l'entorn que vam crear abans, quedant de la següent manera: "$(OPENCV_DIR)\lib". Ara necessitem especificar les

llibreries que volem que el programa vagi a buscar, llavors anem a *Entrada*, que està a la mateixa pestanya que *General*. Al primer apartat anomenat *Dependències addicionals* introduirem el nom de tots el mòduls de la carpeta de llibreries que necessitem.

Els noms de les llibreries segueixen el següent patró:

opencv_(nom del modul)(Versió)d.lib

Exemple: opencv_calib3d2413d.lib

Doncs una vegada fet això haurem de fer tots aquests passos per a *Release | Win32*.

Finalment, si estem treballant amb un ordinador de 64 bits necessitem que el nostre projecte sigui compatible amb aquest, llavors obrim el desplegable que posa *Win32* que està al menú de la capçalera (Fig. 7). Entrem a *Administrador de configuració* i obrim la pestanya de *Plataforma* i fem clic a *Nova...*. Ens assegurem que la nova plataforma és de x64, o sigui 64 bits, i acceptem.

FIG. 7 BARRA DEL MENÚ SUPERIOR

I no ens oblidem de guardar.

Visió artificial

La visió artificial o visió per computació (*computer vision*) és una disciplina científica del camp de la intel·ligència artificial. Inclou mètodes per adquirir, processar, analitzar i comprendre les imatges del món real amb el propòsit de produir informació numèrica que pugui ser tractada per un ordinador. Aquesta disciplina busca automatitzar les tasques que el sistema visual humà realitza, com poden ser el reconeixement facial o comprendre una imatge i prendre decisions.

SIFT

SIFT (*Scale Invariant Feature Transform*) (Alegre & Fernánez-Robles n.d.; Flores & Braun 2011b; Duarte Villaseñor & Chang Fernández 2010; Lindeberg 2012) és un algorisme de visió artificial que serveix per extraure punts característics de les imatges. Aquest algorisme va ser publicat originalment per David Lowe en 1999 i, posteriorment, va ser patentat al 2004 als Estats Units.

Mitjançant aquest punts característics, coneguts com a *keypoints* en anglès, és possible reconèixer l'objecte o figura de la imatge en una base de dades o en altres imatges que contenen més elements. En general, els algorismes de detecció de característiques es basen en la detecció de cantonades, ja que aquestes són invariants a les rotacions, això vol dir que l'algorisme detectarà punts a la cantonada de la figura o de l'objecte encara que aquest estigui sotmès a una rotació o no. Però l'inconvenient d'aquests algorismes és que no treballen bé amb imatges escalades, perquè una cantonada pot deixar de ser una cantonada si la imatge està escalada (Fig. 8). És aquí on SIFT ofereix avantatges, ja que els punts característics extrets amb l'algorisme SIFT són invariants a factors d'escala, translació, rotació i fins i tot, parcialment, a canvis d'il·luminació.

A continuació es descriuran els passos que realitza l'algorisme SIFT.

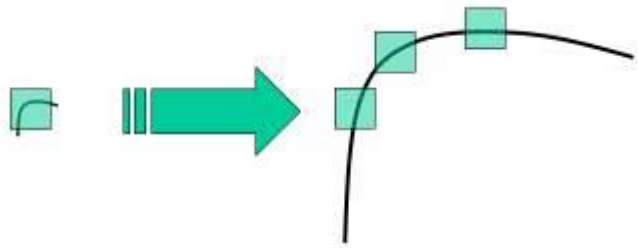

FIG. 8 IMATGE D'UNA CANTONADA AMB EL SEU ZOOM

Detecció d'extrems en l'espai-escala

L'espai-escala (*Scale-space*) d'una imatge consisteix en una família d'imatges derivades, que s'obtenen a partir de la convolució d'una gaussiana de desviació típica σ i escala variable amb la imatge d'entrada.

Espai-escala $L(x, y, \sigma)$;

Escala variable $G(x, y, \sigma)$

Imatge $I(x, y)$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$sabent\ que\ G(x,y,\sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Per aclarir això ens podem fixar en la figura de sota (Fig. 9), on la desviació típica pren diferents valors obtenint imatges cada cop amb menys detall.

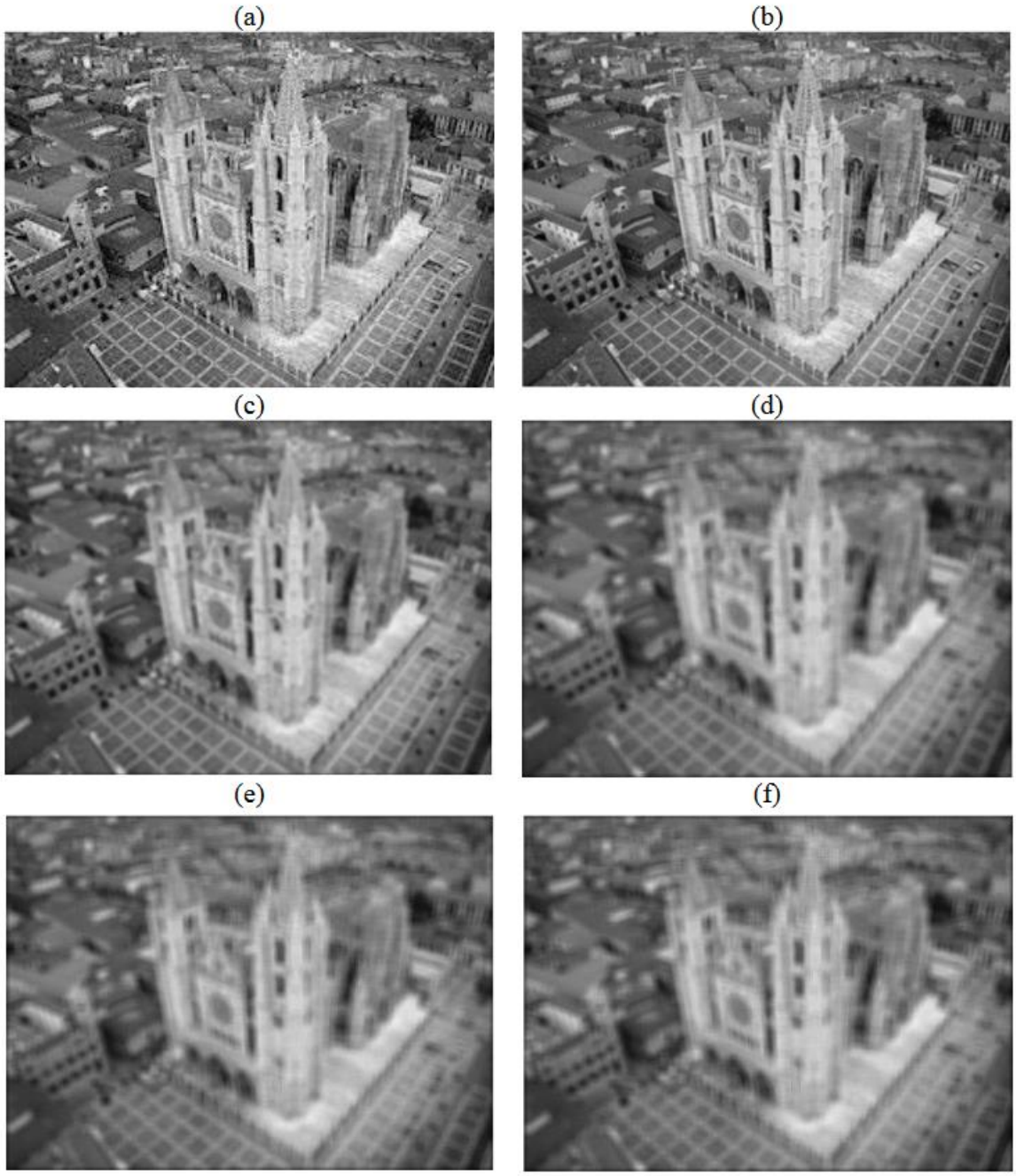

FIG. 9 CATEDRAL DE LLEÓ. (A) σ=0 (IMATGE ORIGINAL); (B) σ=1; (C) σ=2; (D) σ=4; (E) σ=8; (F) σ=16.

Per poder detectar de forma eficient els *keypoints* s'utilitzen els valors extrems (màxims o mínims) de l'espai-escala de la diferència de Gaussianes (DoG) de la imatge, $D(x,y,\sigma)$. Que

es calcula com la diferència de dues escales consecutives separades per un factor constant k que multiplica a la desviació típica.

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

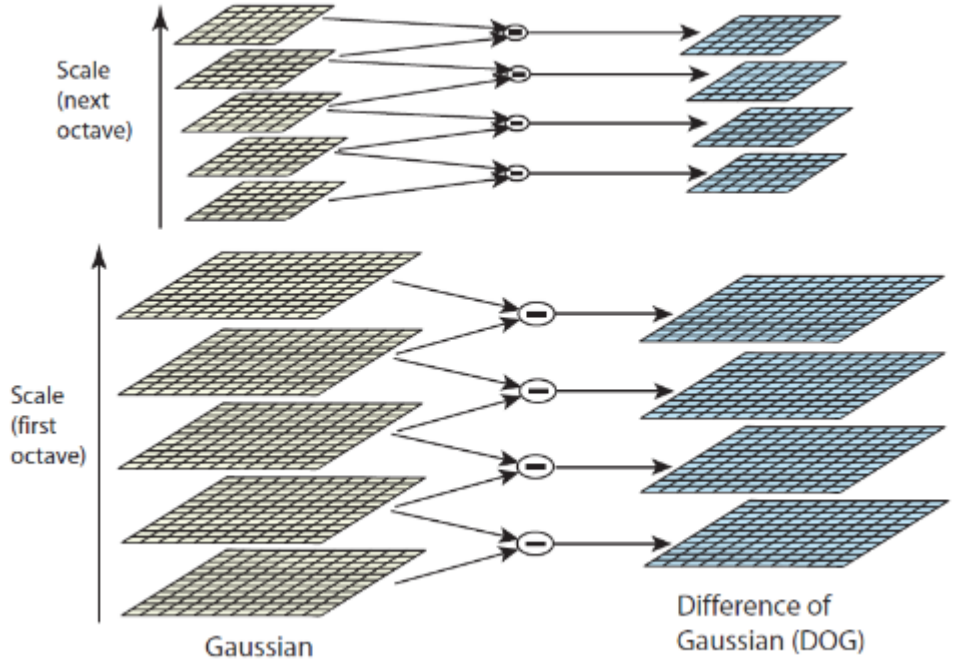

FIG. 10 EXPLICACIÓ GRÀFICA DE L'OBTENCIÓ DE L'ESPAI-ESCALA DE LA DIFERÈNCIA DE GAUSSIANES

Es creen diverses escales reduint successivament la mida de la imatge original, cadascun d'aquests espais-escala que conté les escales s'anomena octava. Cada cop que la mida de la imatge sigui la meitat passem a la següent octava (Fig. 10).

Finalment, per detectar els màxims i mínims locals de la diferència de Gaussianes, $D(x, y, \sigma)$, cada píxel de la mostra es compara amb els seus 8 veïns de la mateixa imatge i també amb els 8 d'una escala inferior i superior com es mostra a la imatge de sota (Fig. 11).

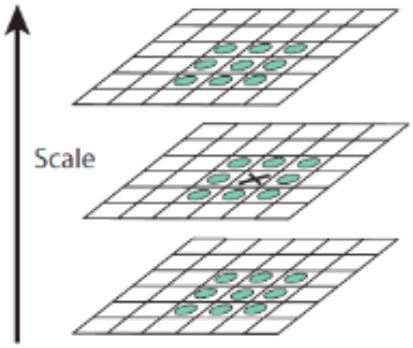

FIG. 11 REPRESENTACIÓ GRÀFICA DEL PÍXEL SELECCIONAT AMB ELS SEUS VEÏNS DELS CORRESPONENTS NIVELLS

Si ens trobem en la transició a la següent octava buscarem els píxels veïns corresponents del nivell superior o inferior. El punt es selecciona com a possible extrem si el valor de D és més gran que els 25 veïns o més petit que aquests.

Localització dels punts d'interès (*Keypoints*)

El pas anterior produeix molts candidats a *keypoints*, però realment molts d'aquests no són bons i s'ha de refinar el llistat de punts. Es procedeix de la següent manera.

Primer s'estima la funció diferència Gaussiana al voltant d'un punt (x_0, y, σ_0) amb una sèrie de Taylor de segon grau.

$$D(X) = D + \frac{\partial D^T}{\partial X^2} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X$$

$$on\ X = (x, y, \sigma)^T\ és\ la\ posició\ relativa$$

Derivant i igualant a 0 obtenim:

$$X' = -\frac{\partial^2 D^{-1}}{\partial X^2} \frac{\partial D}{\partial X}$$

Substituint aquesta última en l'anterior obtenim el valor del màxim local.

$$D(X') = D + \frac{1}{2} \frac{\partial D^T}{\partial X} X'$$

Lowe va proposà un llindar per descartar tots els valors de $|D(X')|$ que fossin més petits del valor 0,03. Llavors, Si $|D(X')| < 0{,}03$ el punt és eliminat de la llista de *keypoints*.

Suposem que D pren valors entre 0 i 1.

A més d'eliminar aquests punts també s'han d'eliminar els punts d'interès obtinguts de les vores de la imatge que es poden confondre amb els de les cantonades del objecte.

La diferència Gaussiana té bona resposta a les voreres de les imatges, per això SIFT utilitza la matriu Hessiana per calcular les curvatures principals.

Llavors utilitzarem la traça i el determinant de la matriu Hessiana (H) de $D(x, y, \sigma)$, també necessitarem els valors propis, on α serà el valor propi de magnitud més gran i β el més petit.

Sabent que:

$$Traça(H) = \frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2} = \alpha + \beta$$

$$Det(H) = \frac{\partial^2 D}{\partial x^2} \times \frac{\partial^2 D}{\partial y^2} = \alpha * \beta$$

Lowe proposa imposar un llindar de r=10 per eliminar els *keypoints* amb una relació de curvatura principal de valor superior a 10.

Considerem r la relació entre els valors propis, sigui α=rβ podem expressar aquesta relació en funció de la traça i el determinant.

$$\frac{Traça(H)^2}{Det(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

Llavors s'imposa la condició següent per poder acceptar el punt d'interès:

$$\frac{Traça(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Assignació d'orientacions

S'assigna a cada *keypoint* obtingut una orientació per garantir la seva invariància respecte a la rotació de la imatge.

Per poder determinar l'orientació correctament a cada *keypoint*, hem de tenir en compte totes les direccions dels punts de la imatge dintre d'un entorn del propi *keypoint*. Llavors es genera un histograma de direccions ponderat per una funció Gaussiana circular centrada al *keypoint* que s'està observant. El màxim de l'histograma correspondrà a la direcció dominant i és la que s'assignarà al *keypoint*. Si existeixen altres màxims amb valor major al 80% del màxim principal, aquests seran utilitzats per crear nous *keypoints* amb aquestes direccions dels màxims en la mateixa posició.

Descriptor de punts d'interès (*keypoints*)

L'últim pas del mètode consisteix en determinar un descriptor per a cada *keypoint*.

Per calcular aquest descriptor, primer s'obté la magnitud i l'orientació del gradient en una finestra de 16x16 píxels centrada en el *keypoint*. Cada regió és dividida en regions de 4x4, i per a cada regió es calcula un histograma de 8 direccions diferents amb les orientacions del gradient. (Fig. 12)

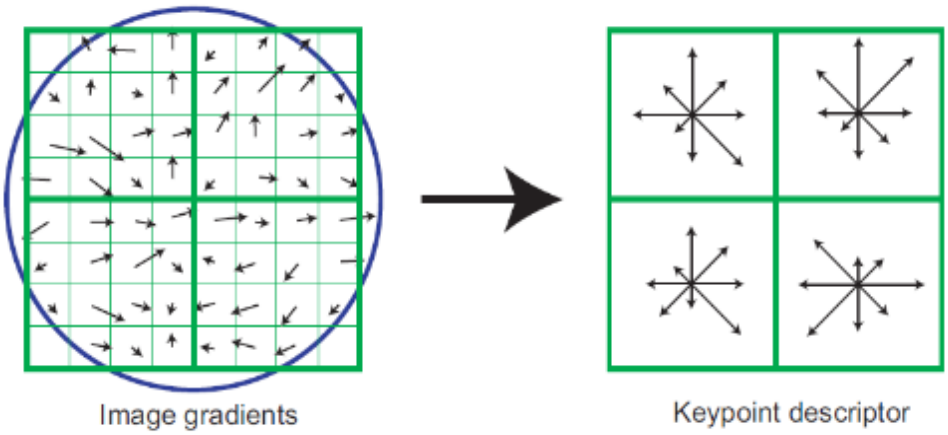

FIG. 12 REPRESENTACIÓ GRÀFICA A ESCALA REDUÏDA. COMENCEM AMB UNA FINESTRA DE 8X8 QUE PASSA A UNA REGIÓ DE 2X2.

Finalment, es concatenen els histogrames de 8 valors i s'obté un descriptor per a cada *keypoint* de 4x4x8=128 valors.

SIFT a la part pràctica

A la part pràctica he obtingut el següent resultat (Fig. 14) on es poden observar els *keypoints* que el programa ha seleccionat. No hi ha cap punt seleccionat fora del que és l'objecte (carta). A l'altre imatge (Fig. 13Fig. 13) s'observen un quants *keypoints* de diferents zones de la carta amb zoom.

FIG. 14 PRIMERA IMATGE AMB KEYPOINTS

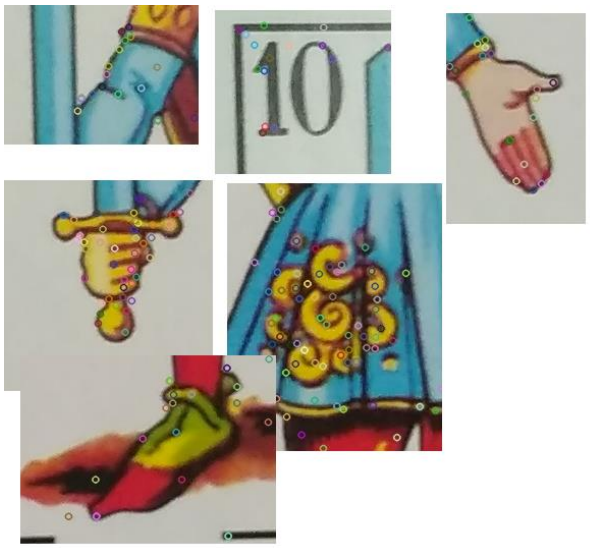

FIG. 13 RETALLS AMB ZOOM DE LA PRIMERA IMATGE

A la segona imatge (Fig. 15) també podem veure tots els *keypoints* i el seu respectiu zoom a diferents zones de la carta (Fig. 16).

FIG. 15 SEGONA IMATGE AMB KEYPOINTS

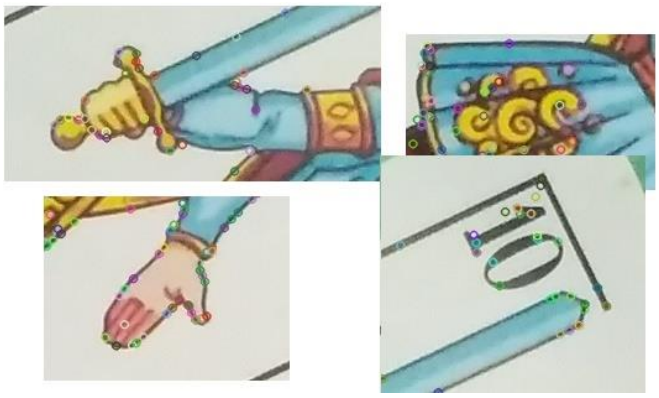

FIG. 16 RETALLS AMB ZOOM DE LA SEGONA IMATGE

Correspondència de punts d'interès (keypoint matching)

Ara que ja tenim els *keypoints* necessitem relacionar els *keypoints* de la primera imatge (imatge objecte) amb els de la segona imatge (imatge escena). Llavors, haurem de buscar

l'homòleg de cada punt a l'altra imatge. Si el punt no té el seu homòleg, aquest punt s'elimina.

OpenCV té implementat 2 algorismes de *matching.* Per una banda tenim l'anomenat *Brute-Force Matcher* i per l'altra banda tenim el *FLANN matcher*.

El *Brute-Force matcher* pren el descriptor d'un *keypoint* i és relacionat amb tots els altres *keypoints* fent servir càlculs de distància. El més proper serà el que serà considerat com a parella d'aquest *keypoint*.

FLANN (*Fast Library for Approximate Nearest Neighbors*) (Bradski 2000) és una llibreria que conté algorismes optimitzats per fer cerques ràpides basades en els veïns més propers en grans conjunts de dades. I utilitza un sistema per triar automàticament el millor algorisme i els paràmetres òptims depenent del conjunt de dades. Aquest mètode és més ràpid que el *Brute-Force*.

Keypoints matching a la part pràctica

A la part pràctica s'ha escollit el mètode FLANN que és més ràpid que el *Brute-Force*.

Com veiem a la imatge (Fig. 17) la relació dels *keypoints* amb els seus homòlegs ha donat un resultat bastant satisfactori. Tots els punts que no tenen homòleg han estat eliminats.

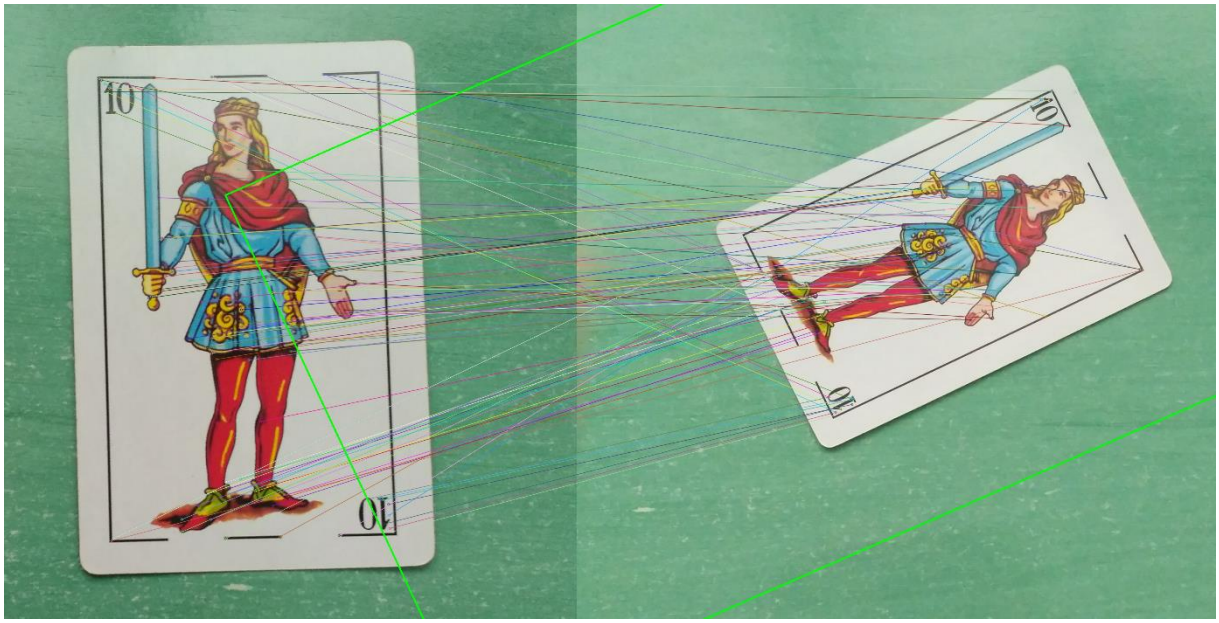

FIG. 17 IMATGE AMB EL MATCHING DELS KEYPOINTS

Malauradament, la zona que enquadra la figura humana és simètrica i això dona alguns problemes. Com podem observar hi ha algun punt de la zona del numero deu o de la línia que enquadra la figura humana que va a parar a la part simètrica de l'altra imatge, ja que aquests són iguals però simètrics.

Algorisme RANSAC

RANSAC (*RANdom SAmple Consensus*)(Flores & Braun 2011a) (Anon n.d.) és un algorisme per a calcular els paràmetres d'un model matemàtic d'un conjunt de dades observades que contenen valors atípics (*outliers*). Va ser públic per primera vegada per Martin L. Fischler y Robert C. Bolles en 1981.

Aquest algorisme donarà millors resultats quan el número d'iteracions sigui gran. Les dades del conjunt consisteixen en *inliers*, és a dir, dades que pertanyen al model. I els *outliers* que són valors que es surten del model. (Fig. 18, Fig. 19)

FIG. 19 CONJUNT DE DADES AMB OUTLIERS

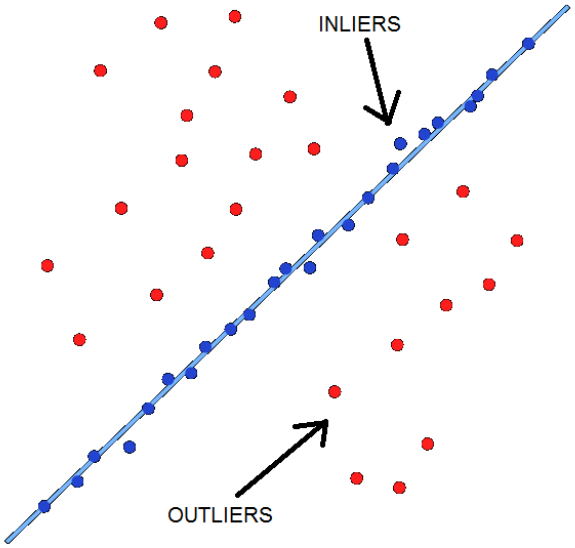

FIG. 18 LÍNIA CREADA AMB RANSAC

L'algorisme es basa en el següent: tenint un conjunt de dades aleatòriament s'escullen 2 punts i es crea una hipòtesi, és a dir, es dibuixa una línia entre els dos punts. A continuació es mesura la distància entre la hipòtesi i cadascun dels punts del conjunt de dades. I tornem

a escollir una altra hipòtesi i a mesurar la distància amb la resta de punts del conjunt, i així successivament. Fins trobar la hipòtesi que tingui la mínima distància fins a la resta de punts, que serà la que millor s'adapti al model. Aquest procés pot ser molt tediós, per estalviar temps s'estableix un llindar de distància a la hipòtesi (Fig. 20), ja que sabem amb certesa que els punts més allunyats de la resta (*outliers*) no formaran part del model.

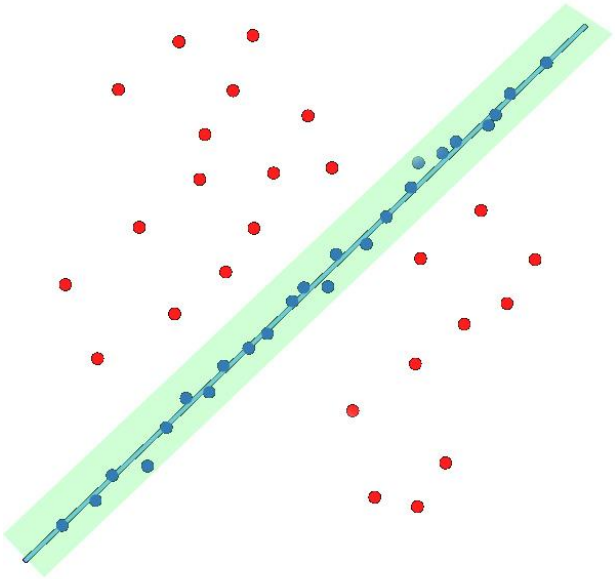

FIG. 20 HIPÒTESI RANSAC AMB UN LLINDAR DE DISTANCIA

RANSAC a la part pràctica

El mètode RANSAC ens permet refinar encara més el llistat de punts obtingut anteriorment, eliminant tots els *outliers*.

En OpenCV està integrat com un paràmetre per calcular la matriu fonamental, la qual s'explica en el següent apartat.

Matriu fonamental

La matriu fonamental (F) (Ma et al. 2004) és una matriu 3x3 que relaciona dues imatges situades al mateix escenari, que es limita a projectar els punts de l'escena en ambdues

imatges. Donada la projecció d'un punt de l'escena en una de les imatges, el punt corresponent a l'altra imatge es relaciona amb una línia, ajudant a la recerca de correspondències equivocades.

La matriu fonamental té les següents propietats.

Per cada parell de punts corresponents x ↔ x' la matriu satisfà:

$$x'^T F x = 0$$

Matriu transposta: si F és la matriu fonamental del parell de càmeres (P, P'), llavors F^T és la matriu fonamental del parell en ordre oposat (P', P).

Línies epipolars: per a qualsevol punt x de la primera imatge, la línia epipolar corresponent és $l' = Fx$. Alhora, $l' = F^T x'$ representa la línia epipolar corresponent a x' en la segona imatge.

El punt epipolar: per un punt x la línia epipolar $l' = Fx$ conté el punt epipolar e'. e' satisfà $e'^T(Fx) = (e'^T F)x = 0$ per a totes les x. Llavors, $e'^T F = 0$ i $Fe = 0$.

La matriu fonamental pot ser calculada sense conèixer els paràmetres interns de la càmera ni la posició relativa.

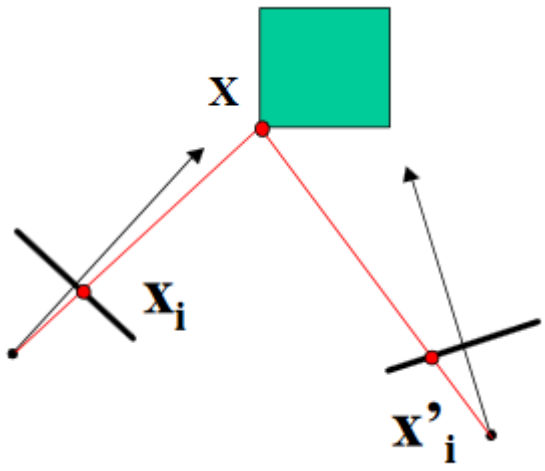

FIG. 21 DOS PUNTS DE DIFERENTES IMATGES QUE CORRESPONEN AL MATEIX PUNT EN L'ESPAI 3D

On Xi i $X'i$ són punts homòlegs a les imatges corresponents. I X és el punt que correspon als dos anteriors en l'espai 3D de l'objecte.

Matriu fonamental a la part pràctica

La matriu fonamental ens servirà, posteriorment, per calcular l'epipolarització de les dues imatges.

En el programa s'executa una ordre que calcula la matriu fonamental de manera ràpida. Només requereix introduir els *keypoints* tant de la primera imatge com els de la segona i seleccionar el mètode pel qual es calcularà, que com s'ha explicat abans s'utilitza el mètode RANSAC.

Epipolarització

Epipolaritzar (Denia Ríos 2011) (Ma et al. 2004) (Camilo et al. n.d.) és simplement passar una línia recta que passi pel punt en la primera imatge i pel punt homòleg d'aquest en l'altre imatge, sense tenir en compte les posicions de les imatges.

La geometria epipolar és la geometria intrínseca entre dues imatges. És independent a la escena, només depèn dels paràmetres interns de la càmera i la posició relativa. Però això ho solucionem utilitzant la matriu fonamental a l'hora de calcular les línies epipolars.

La geometria epipolar entre dues imatges ve donada per la intersecció del pla de cada imatge amb una família de plans, on tots els plans d'aquesta família contenen una línia base que uneix els dos centres òptics de les càmeres.

Aquesta geometria és la fonamental en aplicacions de reconstrucció 3D, ja que s'encarrega de buscar la relació entre les imatges capturades amb un sistema estereoscòpic.

A la següent imatge (Fig. 22) podem veure els elements de la geometria epipolar.

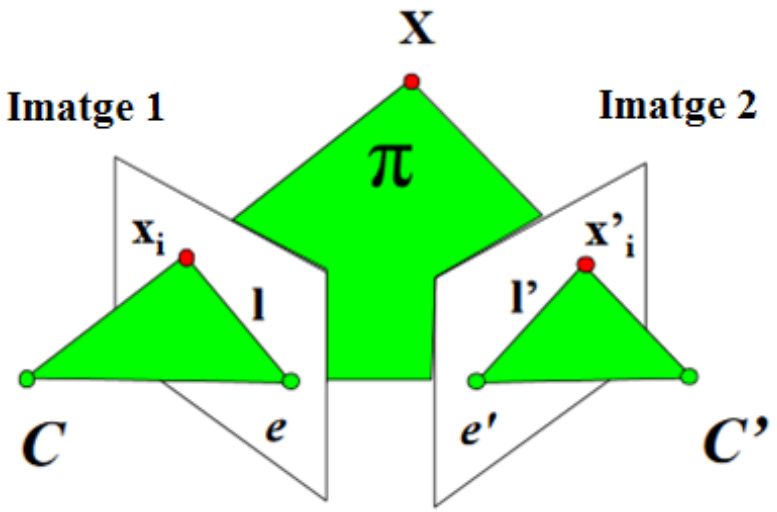

FIG. 22 GEOMETRIA EPIPOLAR

On C i C' són els centres de les càmeres. π és el pla epipolar. X és el punt en l'espai 3D. x i x' són els punts a les imatges, que pertanyen al pla epipolar. e i e' són els epipols i l i l' són les línies epipolars.

Epipol és el punt d'intersecció que uneix els centres (CC') de les càmeres amb el pla de la imatge.

El pla epipolar és el pla que conté la línia base, que uneix els epipols de les càmeres.

Línia epipolar és la línia que intersecta el pla epipolar amb el pla de la imatge. Totes les línies epipolars passen per l'epipol.

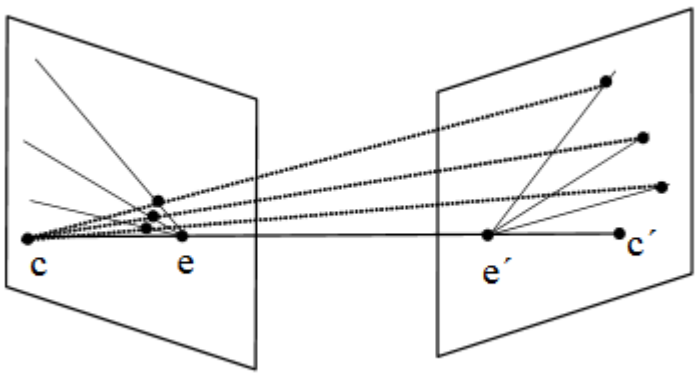

FIG. 23 RELACIÓ DE LES LÍNIES EPIPOLARS ENTRE LES DUES IMATGES

Les línies epipolars estan relacionades amb una transformació projectiva. Llavors existeix una homografia entre les línies epipolars d'una imatge amb l'altra.

Epipolarització a la part pràctica

Per calcular les línies epipolars amb el codi en C++ s'ha utilitzat la instrucció corresponent, la qual utilitza la matriu fonamental. Llavors obtenim el següent resultat per a la imatge 1 la del objecte (Fig. 24).

FIG. 24 IMATGE OBJECTE AMB LÍNIES EPIPOLARS

I per a la segona imatge, la del escenari (Fig. 25) obtenim:

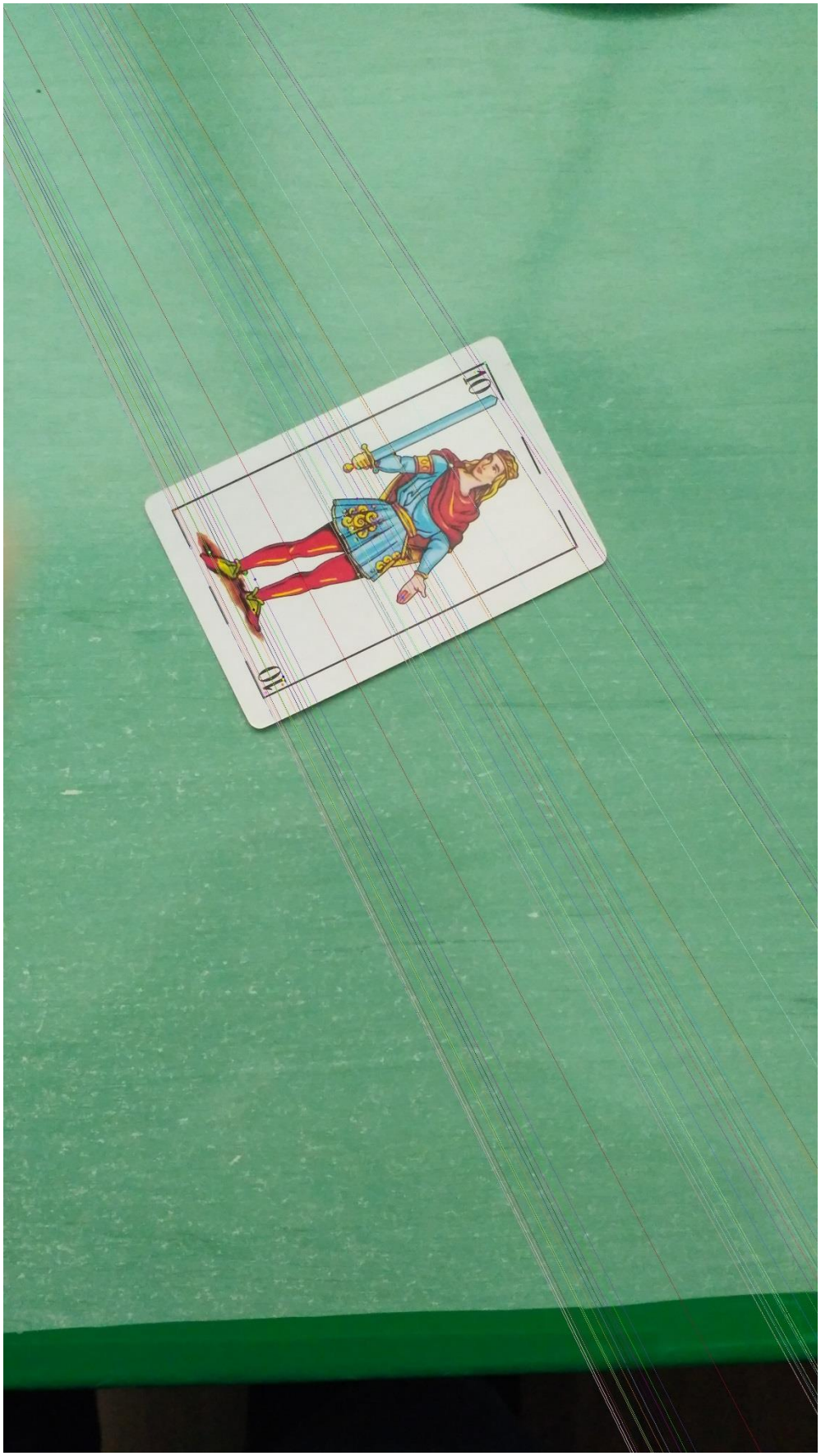

FIG. 25 IMATGE ESCENA AMB LÍNIES EPIPOLARS

Observant les imatges podem dir que el resultat és bastant satisfactori, ja que les línies epipolars passen per les parelles de punts de les imatges. També s'observa que s'han eliminat els problemes de simetria que hi havia quan es va fer el *matching*. Ja que aquests punts que es confonien amb els seus simètrics a l'altra imatge han desaparegut.

Informació addicional

Per fer córrer el programa cal cridar-lo des de la consola d'ordres de Windows (CMD). Per accedir escriurem al buscador d'inici de Windows: "cmd", tal com s'indica a la següent imatge (Fig. 26). I pressionem la tecla *Enter*.

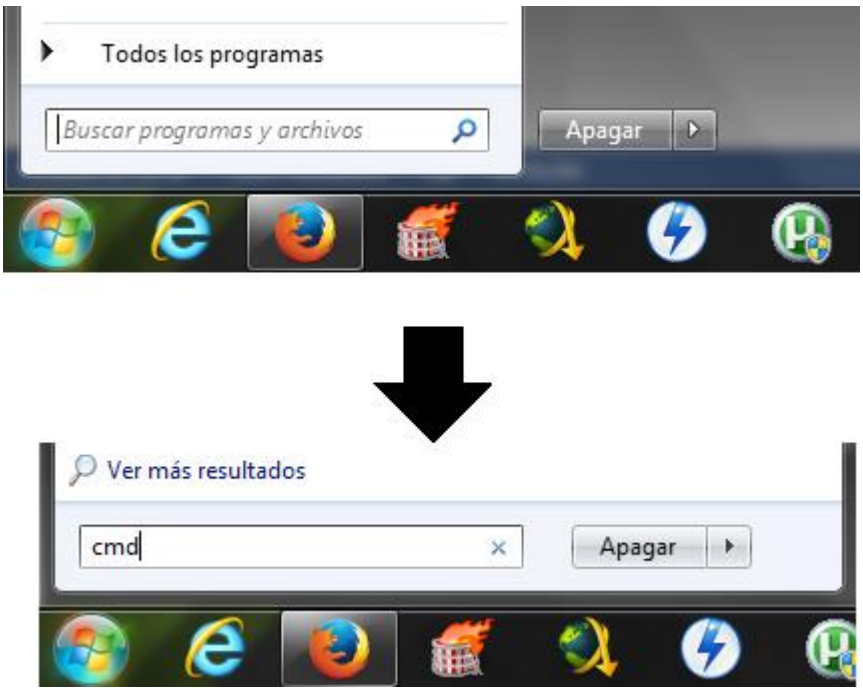

FIG. 26 BUSCADOR DE LA BARRA D'INICI

Depenent de la versió de Windows o del sistema operatiu aquesta interfície pot canviar.

Una vegada a la consola s'ha d'escriure la ruta completa on està ubicat l'arxiu executable del programa, que en aquest cas s'anomena: sift01.exe. Seguit, separat per un espai, els noms de les dues imatges amb les que es vol treballar. Com a l'exemple següent (Fig. 27).

Les dues imatges han d'estar ubicades al mateix directori que l'executable, si no és així haurem d'escriure la ruta completa de cada imatge.

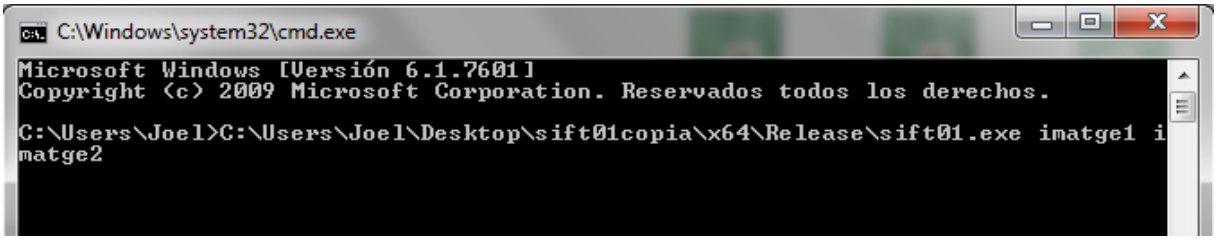

FIG. 27 CMD AMB LA RUTA DEL PROGRAMA

Si a l'hora d'escriure l'ordre ens falten elements el programa no s'executarà i ens mostrarà com fer-ho correctament (Fig. 28).

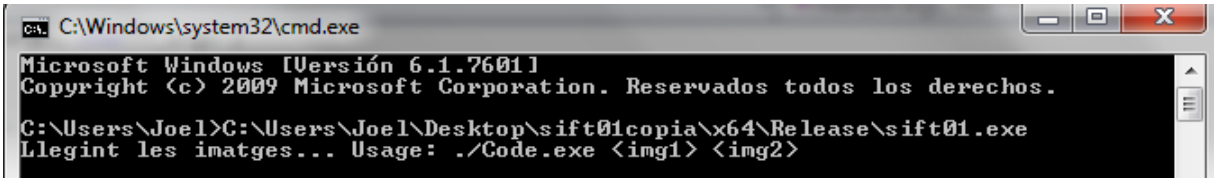

FIG. 28 CMD AMB L'ORDRE INCOMPLETA

Com que al CMD no es pot utilitzar dreceres amb el teclat, haurem de fer clic dret a la barra superior del CMD. Buscar l'opció d'editar i després a enganxar com es mostra a la següent imatge (Fig. 29).

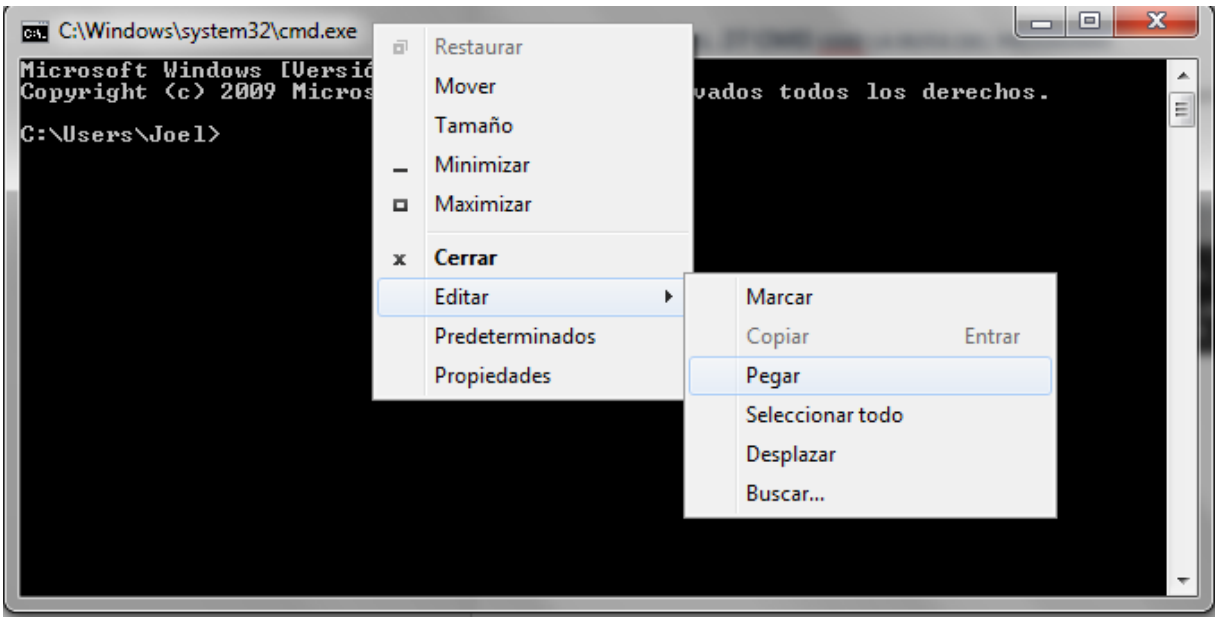

FIG. 29 COM ENGANXAR TEXT AL CMD

Finalment, obtindrem la sortida del programa (Fig. 30) i s'obriran finestres de Windows amb cadascuna de les imatges resultat.

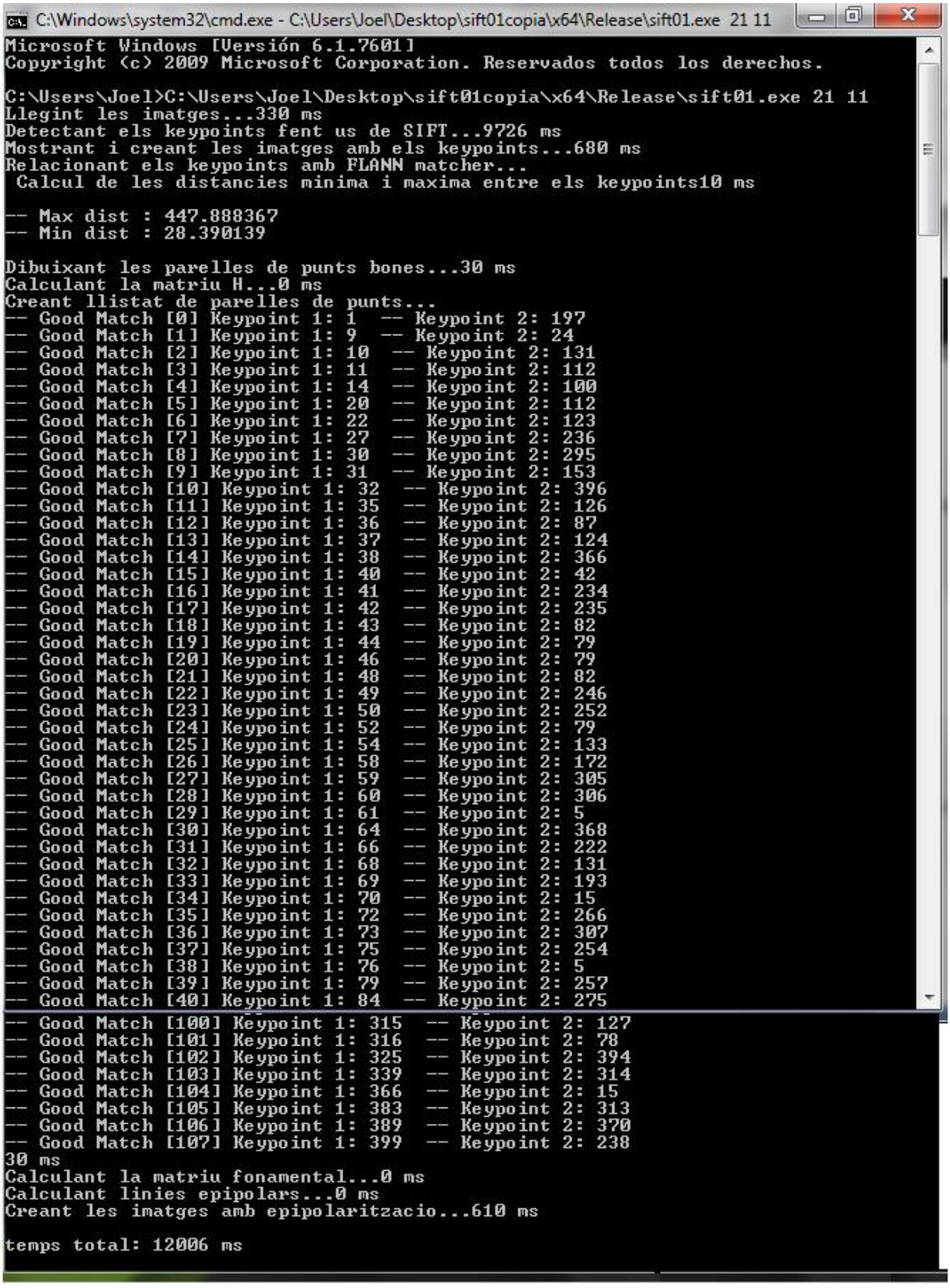

FIG. 30 IMATGE AMB LA SORTIDA DEL PROGRAMA (TALLAT)

Com es pot observar he afegit una instrucció per a que el programa em vagi informant de quant temps trigar a realitzar cada procés. En general, el programa triga entre 11 i 12 segons a executar-se al complet.

Llegir les imatges → 0,3 segons

Detectar keypoints amb SIFT → 9,7 segons

Ensenyar en pantalla i crear imatges amb els keypoints → 0,7 segons

Relacionar els keypoints amb FLANN → 0,01 segons

Dibuixar les parelles de punts → 0,03 segons

Calcular matriu H → menys d'un mil·lisegon

Crear llistat de parelles de punts → 0,03 segons

Calcular la matriu fonamental → menys d'un mil·lisegon

Calcular les línies epipolars → menys d'un mil·lisegon

Crear les imatges amb epipolarització → 0,6 segons

El programa triga una mica a arrencar degut a que el preprocessador ha de llegir unes quantes línies de codi amb *#include* per poder incloure les declaracions d'altres fitxers al compilador.

I com veiem el procés que triga més temps és el de detectar *keypoints* amb SIFT, ja que és més acurat que altres algorisme i ha d'escombrar la imatge sencera en busca de *keypoints*, i empra el mètode explicat a l'apartat de SIFT per a cadascuna de les dues imatges. Els temps emprat a la resta de processos és l'habitual, són processos més curts.

El programa també mostra els resultats que obté en imatges (Fig. 31, Fig. 32, Fig. 33, Fig. 34, Fig. 35). Com que les imatges de sortida s'adapten a la resolució de la pantalla del usuari, es distorsionen i no s'aprecien els detalls molt bé. Per això el codi inclou la instrucció de OpenCV *imwrite* que guarda cadascuna de les imatges que s'obtenen. Són guardades al mateix directori que l'executable.

FIG. 32 IMATGE DE SORTIDA DEL PROGRAMA QUE MOSTRA ELS KEYPOINTS DE LA PRIMERA IMATGE

FIG. 31 IMATGE DE SORTIDA DEL PROGRAMA QUE MOSTRA ELS KEYPOINTS DE LA SEGONA IMATGE

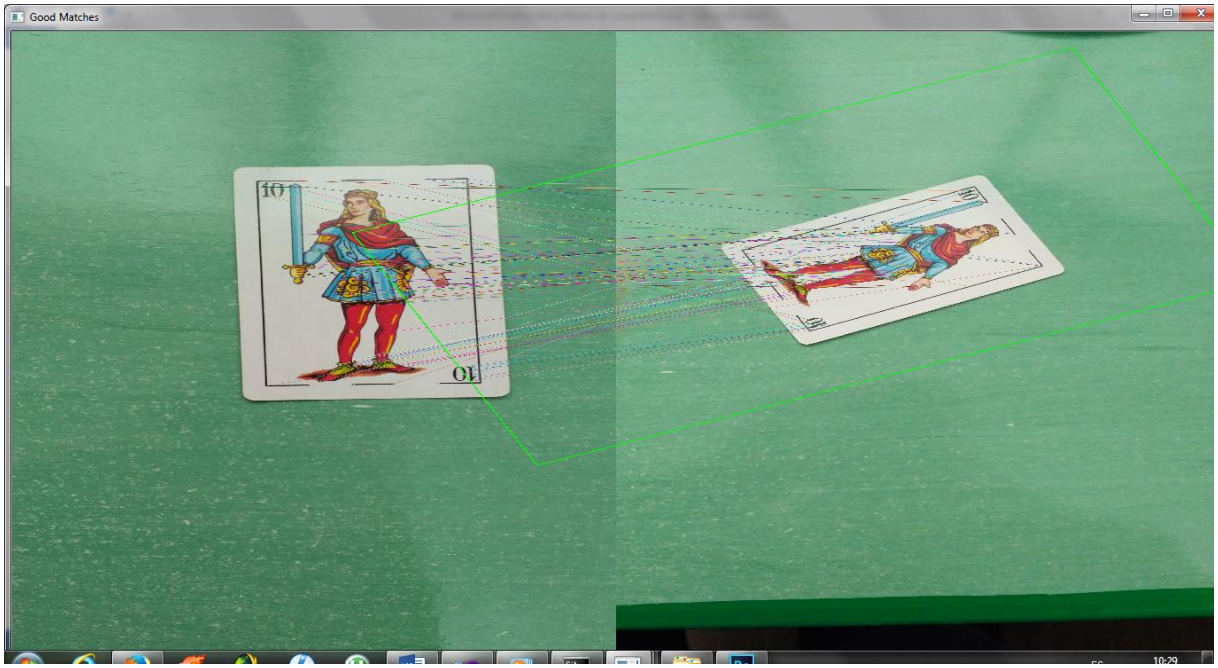

FIG. 33 IMATGE DE LA SORTIDA DEL PROGRAMA QUE MOSTRA EL MATCHING DELS KEYPOINTS

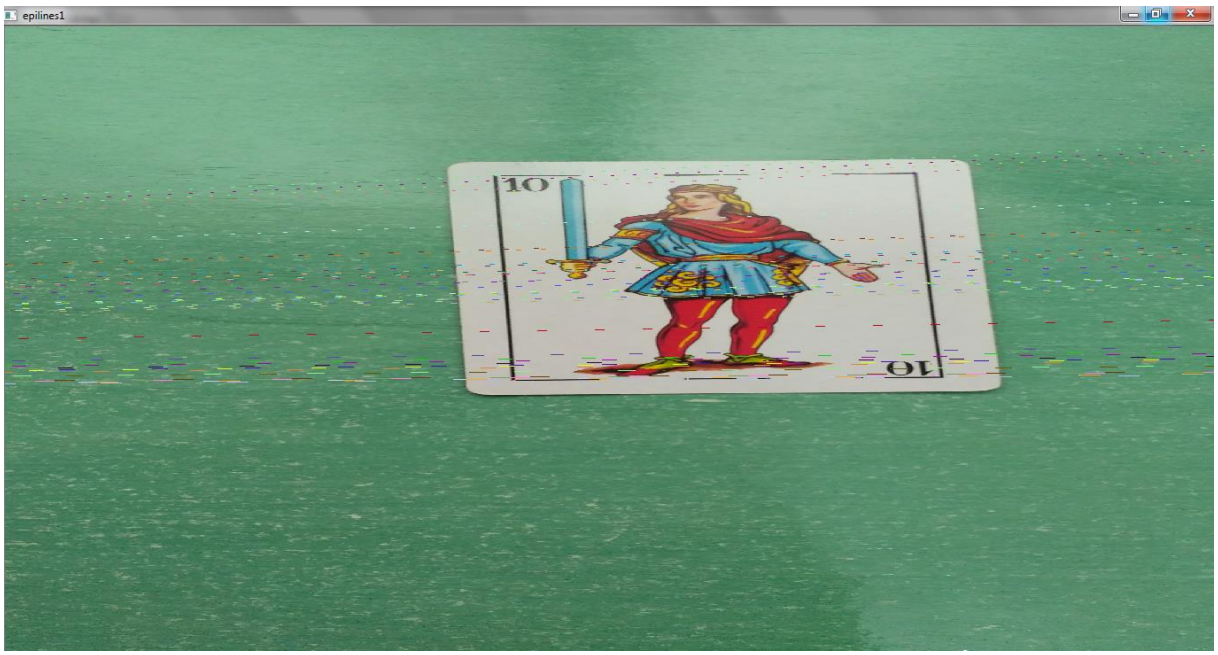

FIG. 34 IMATGE DE SORTIDA DEL PROGRAMA QUE MOSTRA LES LÍNIES EPIPOLARS EN LA PRIMERA IMATGE

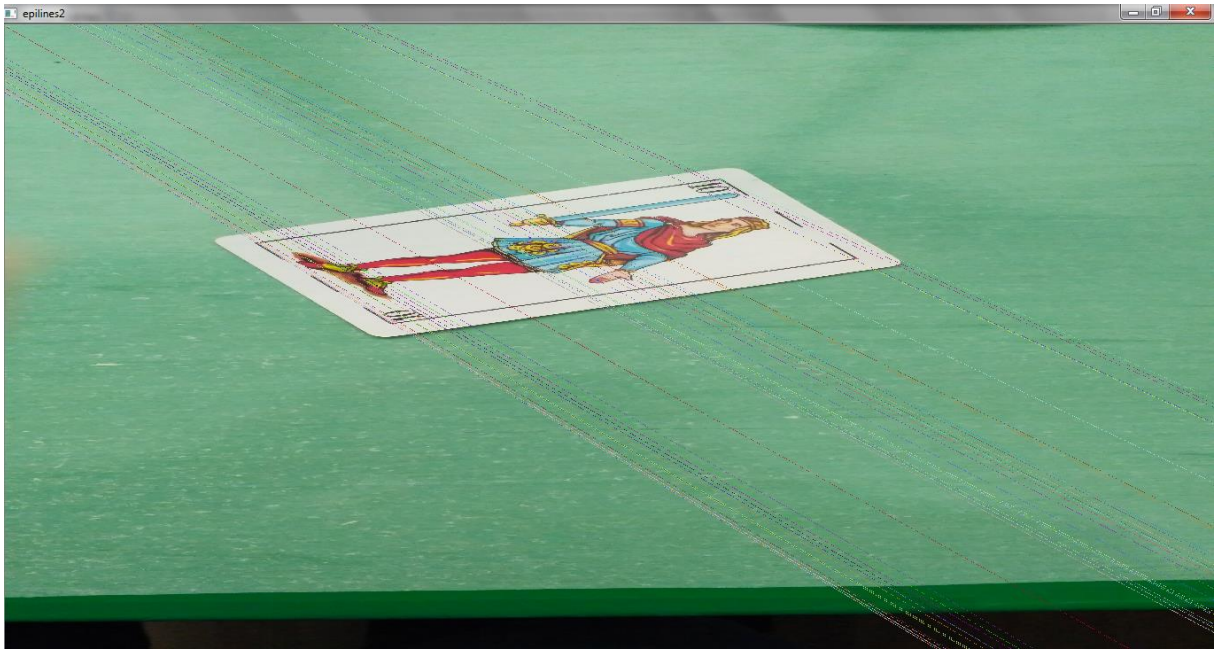

FIG. 35 IMATGE DE LA SORTIDA DEL PROGRAMA QUE MOSTRA LES LÍNIES EPIPOLARS EN LA SEGONA IMATGE

Conclusions

Els resultats obtinguts han sigut bastant bons. He tingut problemes a l'hora d'instal·lar el programa OpenCV i amb la instal·lació de llibreries ja que algunes han donat problemes de compatibilitat ja sigui per l'arquitectura de l'ordinador o pel tipus de sistema operatiu. L'escriptura del codi ha estat en base al mètode de prova i error.

També dir que la simetria per aquests treballs no ajuda gens. Al principi vaig començar utilitzant una carta de pòquer (Fig. 36) i com que la part superior de la carta és totalment simètrica a la part inferior d'aquesta, el propi programa s'embolicava i donava resultats estranys.

FIG. 36 CARTA PÒQUER

Hi ha problemes per automatitzar el procés ja que cada imatge és diferent els paràmetres de les instruccions del codi han de ser diferents, adaptant-se a aquesta imatge. Al fer servir els mateixos paràmetres per a totes les imatges no donaven els resultats òptims que podrien donar si es "personalitza" el codi per a cada parella d'imatges.

L'avantatge d'aquest procés és que podem treballar amb imatges d'origen desconegut.

Bibliografia

Llibres

Gary Bradsky and Adrian Kaehler. Learning OpenCV. 1ª ed. United States of America: O'Reilly Media, 2008. ISBN 978-0-596-51613-0.

Harvey M. Deitel i Paul J. Deitel. C++ how to program. 5ª ed. Nova Jersey: Pearson Education, 2005. ISBN 0131857576.

Hartley, Richard and Andrew Zisserman. Multiple View Geometry in computer vision. 2ª ed. Nova York: Cambridge University Press, 2004. ISBN-10 0-521-540-51-8.

Pàgines web

Alegre, E. & Fernánez-Robles, L., Capítulo 8: SIFT (Scale Invariant Feature Transform). Available at: http://pitia.unileon.es/varp/sites/default/files/PublicationPDF/SIFT (Scale Invariant Feature Transform).pdf.

Anon, RANSAC, SIFT, openCV, Panorama. Available at: http://eric-yuan.me/ransac/.

Bradski, G., 2000. OpenCV library. Available at: http://opencv.org/.

Camilo, A., Flórez Velásquez, A. & Electricista, I., IMPLEMENTACIÓN DE UN ALGORITMO DE CORRESPONDENCIA ENTRE PÍXELES EMPLEANDO GEOMETRÍA EPIPOLAR, PARA APLICACIONES DE VISIÓN ESTEREOSCÓPICA. Available at: http://bibliotecadigital.usb.edu.co/bitstream/10819/3287/1/Implementacion_Algoritmo_Correspondencia_Barros_2015.pdf.

Denia Ríos, J.L., 2011. Epipolarización de un par fotogramétrico, sin parámetros de orientación interior.

Duarte Villaseñor, M.M. & Chang Fernandez, L., 2010. Clasificación de objetos en imágenes usando SIFT. Available at: http://ccc.inaoep.mx/~esucar/Clases-mgp/Proyectos/chang-duarte.pdf.

Flores, P. & Braun, J., 2011a. Algoritmo RANSAC : fundamento teórico. , (2), pp.2–3.

Flores, P. & Braun, J., 2011b. *Algoritmo SIFT : fundamento teórico*, Available at: http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2011/keypoints/FundamentoSIFT.pdf.

Lindeberg, T., 2012. Scale Invariant Feature Transform. , 7(5), p.10491. Available at: http://www.vlfeat.org/api/sift.html.

Ma, Y. et al., 2004. Epipolar Geometry and the Fundamental Matrix. *Multiple View Geometry in Computer Vision*, pp.239–261. Available at: https://www.robots.ox.ac.uk/~vgg/hzbook/hzbook2/HZepipolar.pdf.

Stackoverflow. Stack Overflow - Where Developers Learn, Share, & Build Careers. (2008). Available at: https://www.stackoverflow.com/.

Annexos

Codi escrit en C++

```
#include "stdafx.h"
#include "opencv2/opencv_modules.hpp"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

# include "opencv2/core/core.hpp"
# include "opencv2/features2d/features2d.hpp"
# include "opencv2/highgui/highgui.hpp"
# include "opencv2/calib3d/calib3d.hpp"
# include "opencv2/nonfree/features2d.hpp"
# include "opencv2/imgproc/imgproc.hpp"

FILE *pt;

using namespace cv;

void readme();


int main( int argc, char** argv )
{

clock_t t2, t1, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12, t13, t14, t15, t16, t17,
t18, t19, t20, t21, t22;

t1=clock();
t21=clock();

       printf("Llegint les imatges...");
       if( argc != 3 )
  { readme(); return -1; }

  Mat img_object =
imread("C:\\Users\\Joel\\Desktop\\sift01copia\\x64\\Release\\21.jpg"); //Ruta imatge
del objecte
  Mat img_scene = imread(
"C:\\Users\\Joel\\Desktop\\sift01copia\\x64\\Release\\17.jpg"); //Ruta imatge de
l'escenari

  if( !img_object.data || !img_scene.data )
  { printf(" --(!) Error al llegir les imatges \n"); return -1; }

  t22=clock();
  printf("%d ms", (t22-t21));

  //Detecta els keypoints utilitzant SIFT Detector
  t3=clock();
  printf("\nDetectant els keypoints fent us de SIFT...");
  int minHessian = 400;

  SiftFeatureDetector detector( minHessian );

  std::vector<KeyPoint> keypoints_object, keypoints_scene;
```

```
  detector.detect( img_object, keypoints_object );
  detector.detect( img_scene, keypoints_scene );

  //Calcula els descriptors (feature vectors)
  SiftDescriptorExtractor extractor;

  Mat descriptors_object, descriptors_scene;

  extractor.compute( img_object, keypoints_object, descriptors_object );
  extractor.compute( img_scene, keypoints_scene, descriptors_scene );

  t4=clock();
  printf("%d ms",(t4-t3));

  // Enseña en pantalla els Keypoints
  t5=clock();
  printf("\n");
  printf("Mostrant i creant les imatges amb els keypoints...");
  Mat keyp1;
  drawKeypoints( img_object, keypoints_object, keyp1, Scalar::all(-1),
DrawMatchesFlags::DEFAULT);
  namedWindow("Keypoints1", WINDOW_NORMAL);
  imshow( "Keypoints1", keyp1);
  imwrite( "C:\\Users\\Joel\\Desktop\\sift01copia\\x64\\Release\\Keypoints1.jpg",
keyp1);

  Mat keyp2;
  drawKeypoints( img_scene, keypoints_scene, keyp2, Scalar::all(-1),
DrawMatchesFlags::DEFAULT );
  namedWindow("Keypoints2", WINDOW_NORMAL);
  imshow( "Keypoints2", keyp2);
  imwrite( "C:\\Users\\Joel\\Desktop\\sift01copia\\x64\\Release\\Keypoints2.jpg",
keyp2);

  t6=clock();
  printf("%d ms",(t6-t5));


  //Matching descriptor vectors amb FLANN matcher
  t7=clock();
  printf("\n");
  printf("Relacionant els keypoints amb FLANN matcher...");
  FlannBasedMatcher matcher;
  std::vector< DMatch > matches;
  matcher.match( descriptors_object, descriptors_scene, matches );

  double max_dist = 0;
  double min_dist = 100;

  //Calcul del max i min distancia entre els keypoints
  printf("\n Calcul de les distancies minima i maxima entre els keypoints");
  for( int i = 0; i < descriptors_object.rows; i++ )
  { double dist = matches[i].distance;
    if( dist < min_dist ) min_dist = dist;
    if( dist > max_dist ) max_dist = dist;
  }

  t8=clock();
  printf("%d ms\n\n",(t8-t7));

  printf("-- Max dist : %f \n", max_dist );
  printf("-- Min dist : %f \n", min_dist );
```

```
  //Dibuija només les parelles "bones"
  t9=clock();
    printf("\n");
  printf("Dibuixant les parelles de punts bones...");
  std::vector< DMatch > good_matches;
  vector<Point2f> pts1, pts2;

  for( int i = 0; i < descriptors_object.rows; i++ )
  { if( matches[i].distance < 4*min_dist )
    { good_matches.push_back( matches[i]);
      pts1.push_back( keypoints_scene[matches[i].trainIdx].pt );
      pts2.push_back( keypoints_object[matches[i].queryIdx].pt );}
  }

  Mat img_matches;
  drawMatches( img_object, keypoints_object, img_scene, keypoints_scene,
               good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
               vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );

  t10=clock();
  printf("%d ms",(t10-t9));


  //Localitzant l'objecte de la imatge 1 en la imatge 2
  std::vector<Point2f> obj;
  std::vector<Point2f> scene;

  for( size_t i = 0; i < good_matches.size(); i++ )
  {
    //Obtenir els keypoints dels good matches
    obj.push_back( keypoints_object[ good_matches[i].queryIdx ].pt );
    scene.push_back( keypoints_scene[ good_matches[i].trainIdx ].pt );
  }

  t11=clock();
  printf("\n");
  printf("Calculant la matriu H...");
  Mat H = findHomography( obj, scene, CV_RANSAC, 5 ); // Matriu 3x3

  t12=clock();
  printf("%d ms",(t12-t11));

  //Get the corners from the image 1 ( l'objecte que ha de ser detectat )
  std::vector<Point2f> obj_corners(4);
  obj_corners[0] = Point(0,0); obj_corners[1] = Point( img_object.cols, 0 );
  obj_corners[2] = Point( img_object.cols, img_object.rows ); obj_corners[3] = Point(
0, img_object.rows );
  std::vector<Point2f> scene_corners(4);

  perspectiveTransform( obj_corners, scene_corners, H);


  //Dibuixa les linies entre les cantonades ( the mapped object in the scene - image 2
)
  Point2f offset( (float)img_object.cols, 0);
  line( img_matches, scene_corners[0] + offset, scene_corners[1] + offset, Scalar(0,
255, 0), 4 );
  line( img_matches, scene_corners[1] + offset, scene_corners[2] + offset, Scalar( 0,
255, 0), 4 );
  line( img_matches, scene_corners[2] + offset, scene_corners[3] + offset, Scalar( 0,
255, 0), 4 );
```

```
  line( img_matches, scene_corners[3] + offset, scene_corners[0] + offset, Scalar( 0,
255, 0), 4 );

  //Enseña els matches que són bons
  namedWindow("Good Matches", WINDOW_NORMAL);
  imshow( "Good Matches", img_matches);

  imwrite( "C:\\Users\\Joel\\Desktop\\sift01copia\\x64\\Release\\Matches.jpg",
img_matches);

  //Llistat de parelles bones
  t13=clock();
  printf("\n");
  printf("Creant llistat de parelles de punts...");
  printf("\n");

if
((pt=fopen("C:\\Users\\Joel\\Desktop\\sift01copia\\x64\\Release\\GoodMatches.txt","w")
)==NULL) //reservar memoria// // avisa si no hi ha suficient memoria//
       {
              puts("No hi ha prou espai");
              system("pause");
              exit(0);
       }

  for( int i = 0; i < (int)good_matches.size(); i++ )
  {
       printf( "-- Good Match [%d] Keypoint 1: %d  -- Keypoint 2: %d \n", i,
good_matches[i].queryIdx, good_matches[i].trainIdx );
       fprintf( pt, "-- Good Match [%d] Keypoint 1: %d  -- Keypoint 2: %d \n", i,
good_matches[i].queryIdx, good_matches[i].trainIdx );
  }

  fclose(pt);

  t14=clock();
  printf("%d ms",(t14-t13));

  // Calcular matriu fonamental (Matriu 3x3)
  t15=clock();
  printf("\n");
  printf("Calculant la matriu fonamental...");
  Mat F = findFundamentalMat(obj, scene, CV_FM_RANSAC, 3, 0.99);

  t16=clock();
  printf("%d ms",(t15-t16));

  // Epipolarització
  t17=clock();
  printf("\n");
  printf("Calculant linies epipolars...");
  vector<Vec<float,3> > epilines1, epilines2;
computeCorrespondEpilines(obj, 1, F, epilines2);
computeCorrespondEpilines(scene, 2, F, epilines1);


CV_Assert(obj.size() == scene.size() &&
        obj.size() == epilines1.size() &&
        epilines1.size() == epilines2.size());

t18=clock();
printf("%d ms",(t18-t17));
```

```
t19=clock();
printf("\n");
  printf("Creant les imatges amb epipolaritzacio...");

cv::RNG rng(0);

for(unsigned int i=0; i<obj.size()/2; i++)
{
cv::Scalar color(rng(256),rng(256),rng(256));
     line(img_object, Point(0,-epilines1[i][2]/epilines1[i][1]),
Point(img_object.cols,-
(epilines1[i][2]+epilines1[i][0]*img_object.cols)/epilines1[i][1]),color);
     circle(img_object, obj[i], 3, color, -1, CV_AA);

     line(img_scene, Point(0,-epilines2[i][2]/epilines2[i][1]), Point(img_scene.cols,-
(epilines2[i][2]+epilines2[i][0]*img_scene.cols)/epilines2[i][1]),color);
     circle(img_scene, scene[i], 3, color, -1, CV_AA);
}


namedWindow("epilines1", WINDOW_NORMAL);
imshow("epilines1", img_object);

imwrite( "C:\\Users\\Joel\\Desktop\\sift01copia\\x64\\Release\\epilines1.jpg",
img_object);


namedWindow("epilines2", WINDOW_NORMAL);
imshow("epilines2", img_scene);

imwrite( "C:\\Users\\Joel\\Desktop\\sift01copia\\x64\\Release\\epilines2.jpg",
img_scene);

t20=clock();
printf("%d ms",(t20-t19));

t2=clock();
printf("\n\ntemps total: %d ms",(t2-t1));



  waitKey(0);

  return 0;
}


void readme()
{ printf(" Usage: ./Code.exe <img1> <img2>\n"); }
```

Traducció del 30% del nucli del TFG

What's OpenCV?

OpenCV (Open Source Computer Vision Library) is a group of libraries created by Intel, designed to computate applications in real time. It has over 2,500 optimized algorithms, which can be used in different ways: detection and recognition of human faces, identification of objects, tracking of movements from objects, finding similar images in an image data base, etc.

OpenCV is free for academic or commercial purposes. It is used with the BSD licence (Berkeley Software Distribution). Platforms, such as Windows, Linux, Max OS, IOS and Android are supported. Originally, it was written in C++, but it can also be used in Phython, Java, C, C++ and MATLAB.

Instal·lació OpenCV

Firstly, we need to download and install Microsoft Visual Studio, concretely Microsoft Visual Studio 2010. Such software will allow us to write the code.

Next, we must download OpenCV from the official website (www.opencv.org), with which we will obtain a series of libraries that will permit us work with images.

In this case, version 2.4.13 was used because of the fact that all the modules were included in the installer. However, at the most recent version (3.2.0) not all modules have the installer, actually there are libraries that are needed known as *nonfree* that must be installed apart.

Once it is downloaded, we have to extract the file and place it on the directory we want, preferably we should put it into the root of the hard drive that tends to be: *"C:\"*, since it's a short path and will help us to avoid problems.

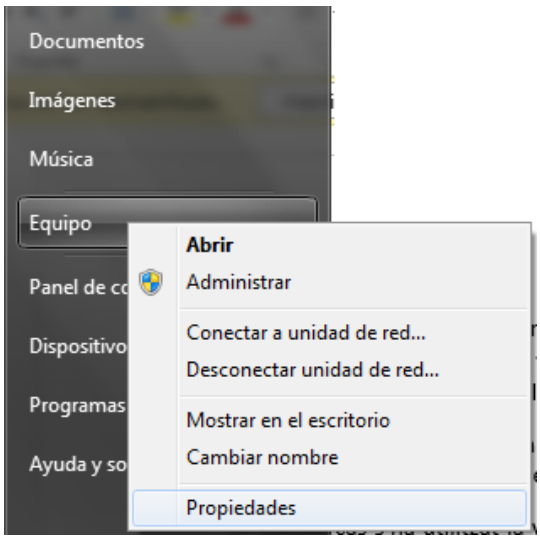

FIG. 1 PC PROPERTIES

Then, after heading to the PC properties (Fig. 3), we must access to the *"Advanced System configuration*" and then to the *"Environment variables"*.

After that, we will create the new variables clicking "New...". Depending if the software is used by just one user or by the whole system, we will create the variable in one of the options. Then, we need to write OPENCV_DIR as the variable name, underneath the path of the binaries files and libraries of OpenCV, like the picture that we can see below (Fig. 4).

We have to keep in mind the architecture of our operative system when it comes to writing the path. We will write x64 if we have a 64 bits' operative system or x86 in case it is a 32 bits' operative system.

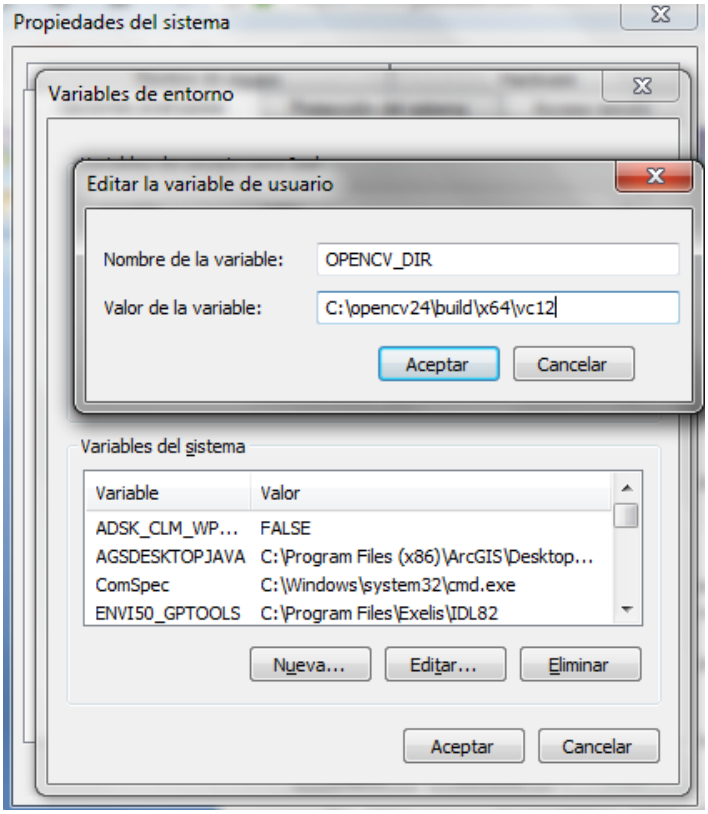

FIG. 2 ENVIRONMENT VARIABLES

At the section of Variables of the system we have to edit the variable *PATH* so that we add the file path named bin of OpenCV, which, in my case, is the following: ";%OPENCV_DIR%\bin". ";" is written at the beginning to split the patches that the variable contains.

Once we have the software installed, we need to configure the Microsoft Visual Studio, which, in my case, is the 2010 version (Microsoft Visual Studio 2010).

Thereon, we open the Microsoft Visual Studio and we create a new project, concretely, a console application of Win32. Following on, we head to the left menu known as Properties administrator.

First, we will create a template for the project priorities. This way every time we create a new project in which we want to use OpenCV we will only need to load our template of properties that will contain all the modifications made at the beginning.

Therefore, we will right click our project's name so that we can have access to *Add new properties sheep,* we will name it and then save it. As previously mentioned, this template will keep all the changes that we make to the properties.

La propera vegada que vulguem obrir un nou projecte per a treballar amb OpenCV només haurem d'anar a la tercera opció de la barra de l'*Administrador de propietats* com s'indica a la imatge de sota. I carregar la nostra plantilla, per no fer cada vegada que obrim un nou projecte el que ve a continuació.

The next time that we want to open a new project to work with OpenCV we will only need to go to the third option from the bar of the *Properties administrator* like we can see at the picture below (Fig. 3). In addition, we will have to load our template in order to avoid doing the next steps every time we open a new project.

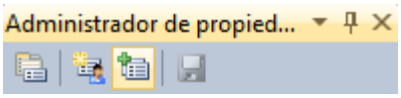

FIG. 3 PROPERTIES ADMINISTRATOR BAR

After having the template ready, we will right click *Debug | Win32* so that we can open the properties and we will modify some options in order to be able to make use of OpenCV inside this software. We will open the *C/C++* tab and go to *General*, at the first section we will write the file path *Include* of OpenCV, which, in this case, is the following: "C:\opencv24\build\include".

Right after, we will go to the *Linker* tab and access to *General*, at the section of *Additional libraries directory* we will introduce the file path that contains the OpenCV libraries. In this case we can utilize the variable of the environment that we created previously, resulting this way: "$(OPENCV_DIR)\lib". After that, we will need to specify the libraries that we want the software to look for, then we will go to the *Input,* which is at the same tab as *General.* At the

first section known as *Additional dependency* we will introduce the name of all the modules of the libraries file that we need.

The libraries names follow this order:

opencv_(nom del modul)(Versió)d.lib

Example: opencv_calib3d2413d.lib

Once all this is done, we will have to follow all these steps for *Release | Win32*.

Lastly, if we are working with a 64 bits computer we will need that our project is compatible with this one. Then, we will open the open-down identified as *Win32* which is located at the top menu. We will access to *Configuration administrator* and will open the *Platform* tab and we will click *New….* We will make sure that the new platform is a x64 platform, that is to say 64 bits, and we will click *Accept*.

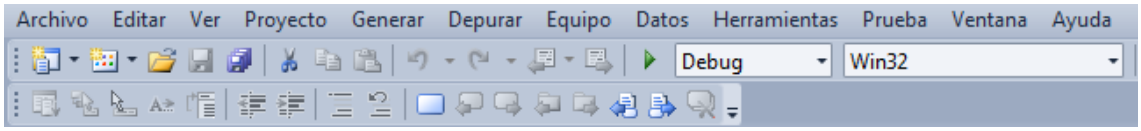

FIG. 4 TOP MENU BAR

Important reminder: save.

Computer vision

The artificial vision or computer vision is a scientific discipline of the artificial intelligence field. It includes methods to acquire, process, analyze and understand the images of the real world with the aim of producing numeric information that can be treated by a computer. The discipline's purpose is automating the tasks that the human visual system does, such as the facial recognition or understanding an image and take decisions.

SIFT

SIFT (Scale Invariant Feature Transform) is algorithm of artificial vision that is used to extract characteristic points of the images. Such algorithm was originally published by David Lowe in 1999 and subsequently it was patented in 2004 in the US.

By means of this characteristic points, known as *keypoints*, it is possible to recognize the object or the image figure in a data base or on other images which contain more elements. Generally, the algorithms of detection of characteristics are based in the detention of corners due to its invariable rotations, which means that the algorithm will detect the points of the corner of the figure/object although it is or isn't subdued to a rotation. However, the disadvantage of these algorithms is the fact that they do not properly work with escalated images because a corner can stop being a corner if the image is escalated. It's here where SIFT offers advantages as the characteristic points are extracted with the SIFT algorithm are invariable in scale factors, translation, rotation and even in illumination changes partially.

Next, the steps that the SIFT algorithm makes will be described.

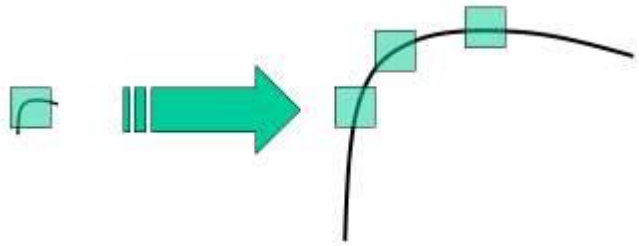

FIG. 5 IMAGE OF A CORNER WITH HIS OWN ZOOM IN

Scale-space extrema detection

The *Scale-space* of an image consists in a family of derived images, which are obtained thanks to the Gaussian convolution of the standard deviation σ and a variable scale with the input image.

Scale-space $L(x, y, \sigma)$;

Variable scale $G(x, y, \sigma)$

Image $I(x, y)$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$Knowing\ that\ G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

In order to clarify it, we can look at the below figure (Fig. 9) in which the standard deviation takes different values obtaining images less detailed images every time.

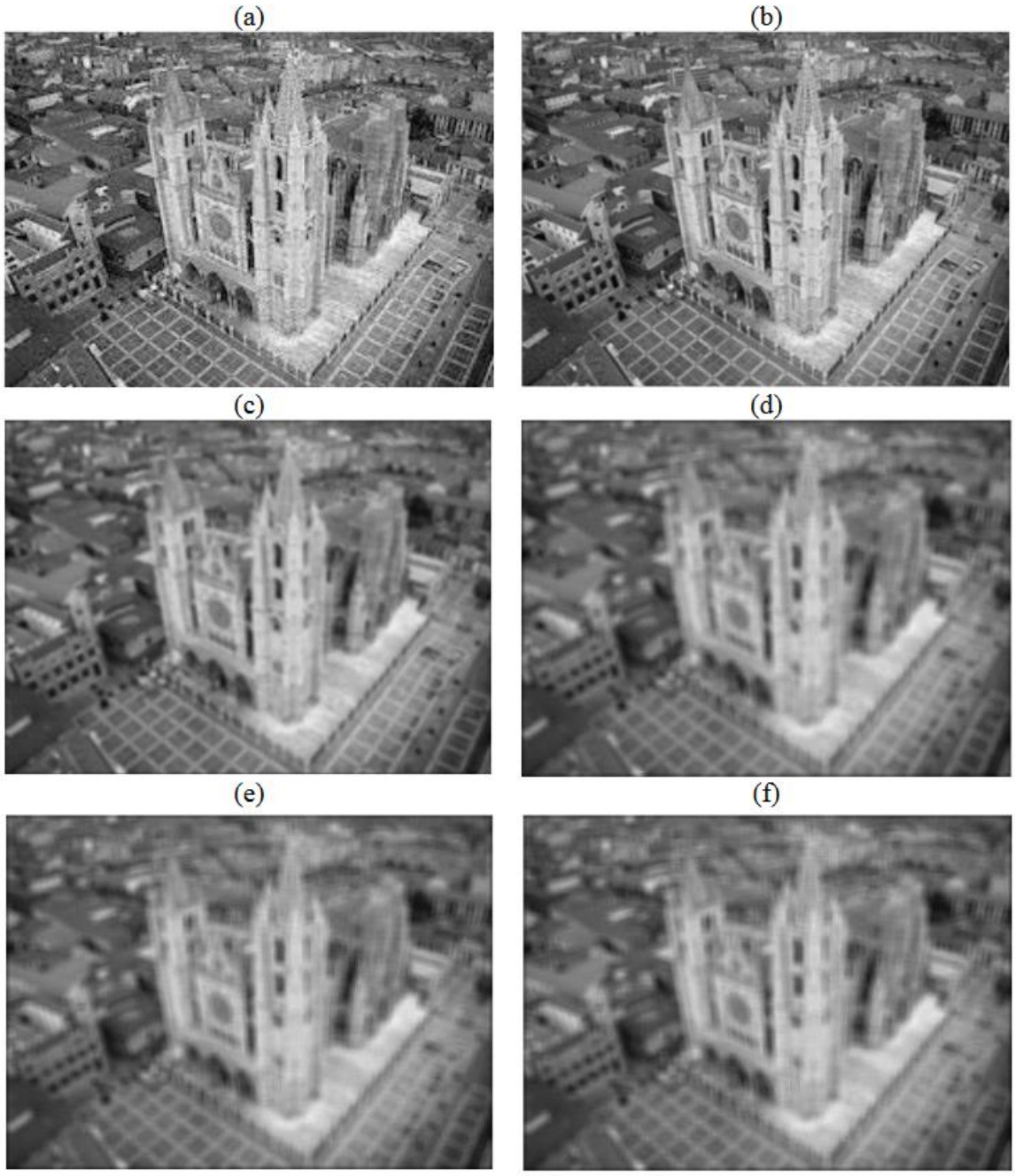

FIG. 6 CATEDRAL DE LLEÓ. (A) Ϭ=0 (ORIGINAL PICTURE); (B) Ϭ=1; (C) Ϭ=2; (D) Ϭ=4; (E) Ϭ=8; (F) Ϭ=16.

In order to efficiently detect the keypoints, the extrema values (maximum or minimum) of the scale-space of the difference of Gaussian (DoG) of the image, $D(x, y, \sigma)$. It is calculated as the difference of two consecutive scales splited by a constant factor *k* that multiplies the standard deviation.

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

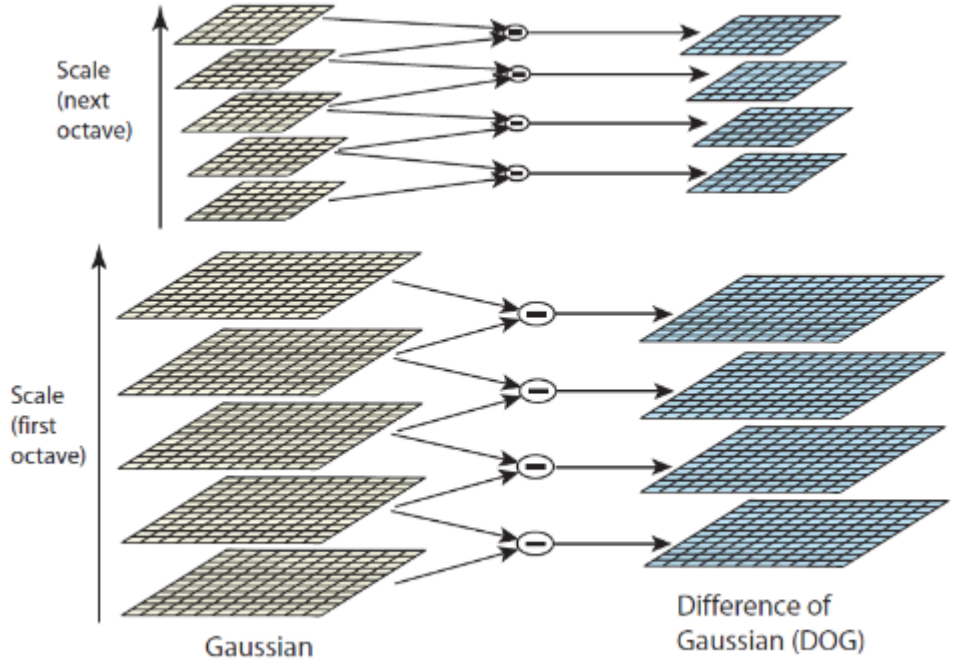

FIG. 7 GRAPHIC EXPLANATION OF OBTAINING THE SPACE-SCALE OF DOG

Several scale are created successively reducing the dimensions of the original image, each of the scale-spaces that the scales contain are known as octave. Each time the images dimensions are the half we go over to the following octave.

Finally, in order to detect the local maximums and minimums of the difference of Gaussian, $D(x, y, \sigma)$, each pixel of the sample is compared to its 8 neighbours of the same image and also to the 8's of an inferior and superior scale as we can see in the image below (Fig. 11).

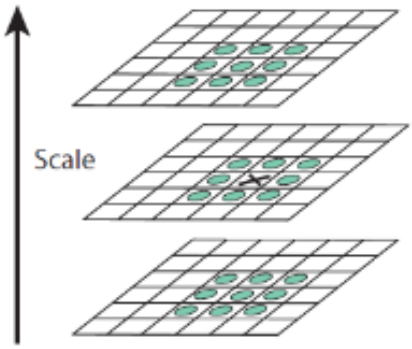

FIG. 8 GRAPHIC REPRESENTATION OF THE SELECTED PIXEL AND ITS NEIGHBORS

If we are working on the transition of the next octave, we will search the corresponding neighbour pixels of the superior or inferior level. The point must be selected as a possible extrema in case the D value is bigger than the 25 neighbours or smaller than these ones.

Keypoints location

The previous step produces many candidates to keypoints. Nevertheless, actually many of these are not correct so we have to refine the list of points. It is done in the following way.

First of all, the difference Gaussian function is estimated around the point (x_0, y, σ_0) with a series of Taylor of second degree.

$$D(X) = D + \frac{\partial D^T}{\partial X^2} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X$$

$$on\ X = (x, y, \sigma)^T\ is\ the\ relative\ position$$

Derivating and matching the results to 0 we obtain:

$$X' = -\frac{\partial^2 D^{-1}}{\partial X^2} \frac{\partial D}{\partial X}$$

By replacing these last one on the previous one we obtain the maximum local value.

$$D(X') = D + \frac{1}{2} \frac{\partial D^T}{\partial X} X'$$

Lowe proposed a threshold to discard all the $|D(X')|$ values that were smaller than 0,03. Then, if $|D(X')| < 0{,}03$ the point is deleted from the keypoints list.

Let's suppose that *D* takes values between 0 and 1.

Apart from deleting these points, the points of interest obtained of the corners of the image have to be deleted as well, considering that they could be mixed up with the ones located at the object's corners.

The difference of Gaussian has a good response to the corners of the image. Therefore, SIFT uses the Hessian matrix to work the main curvatures out.

Then, we will use the trace and the determinant of the Hessian matrix (H) of $D(x, y, \sigma)$, also we will need the proper values, α will be the biggest of the magnitude value and β smallest.

Considering that:

$$Trace(H) = \frac{\partial^2 D}{\partial x^2} + \frac{\partial^2 D}{\partial y^2} = \alpha + \beta$$

$$Det(H) = \frac{\partial^2 D}{\partial x^2} \times \frac{\partial^2 D}{\partial y^2} = \alpha * \beta$$

Lowe proposed imposing a r=10 threshold in order to delete the keypoints with a relation of principal curvature of value higher than 10.

r is considered the relation between the proper values, α=rβ this relation can be expressed depending on the trace and the determinant.

$$\frac{Trace(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

Then, the following condition is imposed in order to accept the point of interest:

$$\frac{Trace(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Orientations assignation

Each obtained keypoint is assigned to an orientation for guaranteeing its invariability regarding the rotation of the image.

For the purpose of properly determining the orientation to each keypoint we ought to have in mind all the direction of the image's points inside the environment of the keypoint itself. Then, we need to get a histogram of directions weighted by a circular Gaussian centred at the keypoint that we can see.

The maximum of the histogram will correspond to the dominant direction and will be assigned a keypoint. If other maximums of the value bigger than the 80% of the maximum principal exist, these will be used to create new keypoints with these directions of the maximums at the same position.

Keypoints descriptor

The last step of the method consists on determining a descriptor for each keypoint.

To work such descriptor out, firstly, we need to obtain the magnitude and the gradient orientation in a 16x16 window centered in the keypoint. Each region is divided into 4x4 regions, and a histogram of 8 different directions with gradient orientations needs to be worked out. (Fig. 9)

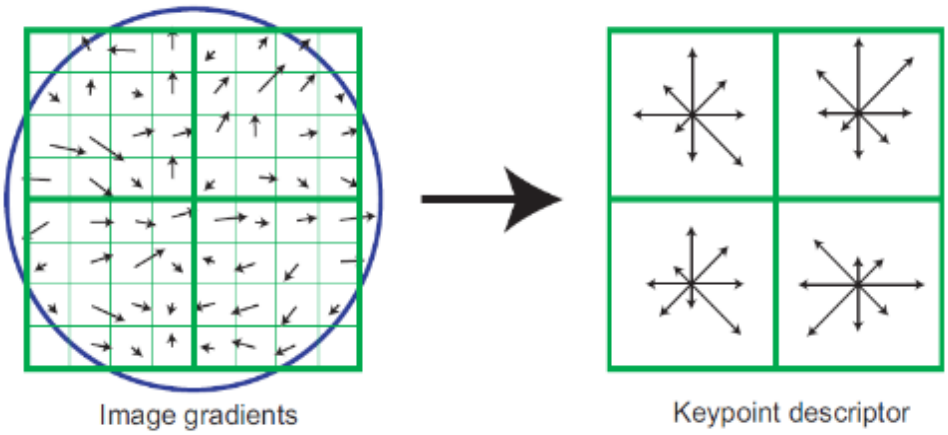

FIG. 9 GRAPHIC REPRESENTATION

Lastly, the histograms of 8 values are concreted and a descriptor is obtained for each keypoint of 4x4x8=128 values.

SIFT in the practice

At the practical part I obtained the following result (Fig.13) where we can see the keypoints which the program has selected. There is no point selected outside of the object (card). In the other picture (Fig. 14) several keypoints from different areas of the card can be seen by zooming.

FIG. 11 FIRST PICTURE WITH KEYPOINTS

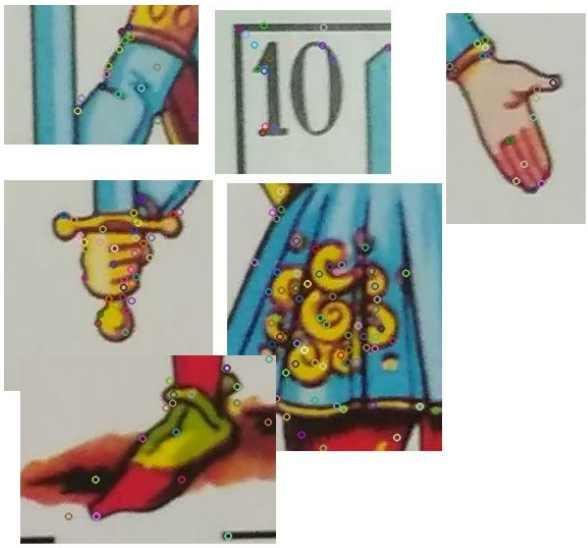

FIG. 10 ZOOM IN CLIPPINGS OF THE FIRST PICTURE

In the second photograph (Fig. 15) we can also see all the keypoints and their own zoom at different areas of the card (Fig. 16).

FIG. 12 SECOND PICTURE WITH KEYPOINTS

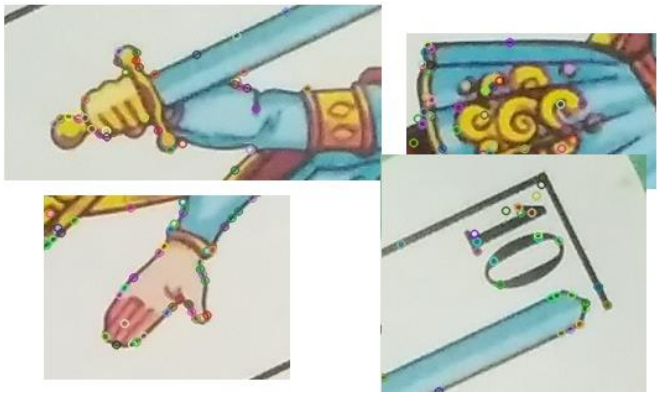

FIG. 13 ZOOM IN CLIPPING OF THE SECOND PICTURE

Keypoint matching

Once we have the keypoints, we need to match the keypoints of the first image (object image) with the keypoints from the second image (scene image). Then, we will need to look for the homologous of each point to the other picture. If the point does not have its homologous, this point has to be deleted.

OpenCV has 2 implemented matching algorithms. On the one hand we have the Brute-Force Matcher and in the other we have the FLANN matcher.

The Brute-Force matcher take the keypoint descriptor and it is matched with all other keypoints by means of distance calculation. The closest one will be the pair of this keypoint.

FLANN (Fast Library for Approximate Nearest Neighbors) (Bradski 2000) is a library that contains optimized algorithms to do fast searches based on the closest neighbors in big groups of data. It uses a system to automatically choose the best algorithm and the optimal parameters depending on the group of data. Such method is faster than the Brute-Force's.

Keypoints matching in the practice

At the practice the FLANN method has been chosen because it is faster than Brute-Force.

We can see in the picture (Fig. 17) the keypoints matching with their homologous that shows a quite satisfactory result. All the points with no homologous has been deleted.

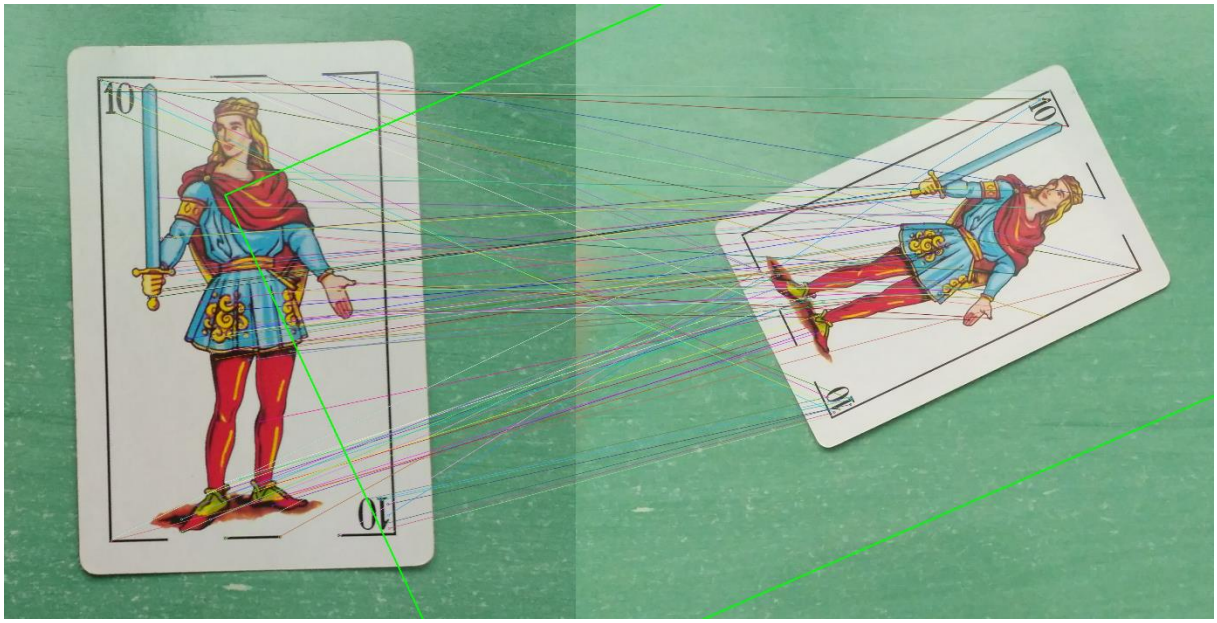

FIG. 14 PICTURE OF THE KEYPOINTS MATCHING

Unfortunately, the zone that frame the human figure is symmetric and this causes some problems. Like we can see there is some points in the number ten zone or on the line that

frames the human figure that goes to the symmetric part of the other image, because those are equal but symmetrical.

RANSAC algorithm

RANSAC (RANdom SAmple Consensus)(Flores & Braun 2011a) (Anon n.d.) is an algorithm to calculate the parameters of a mathematical model of a dataset observed which contains outliers. It was published for the first time by Martin L. Fischler y Robert C. Bolles in 1981.

This algorithm will give better results with a bigger number of iterations. The dataset consist in inliers, rather, data that belongs to the model. And the outliers are values that are outside of the model (Fig. 18, Fig. 19).

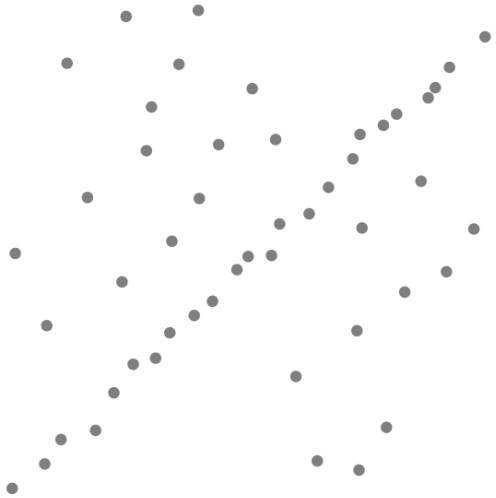

FIG. 16 DATASET WITH OUTLIERS

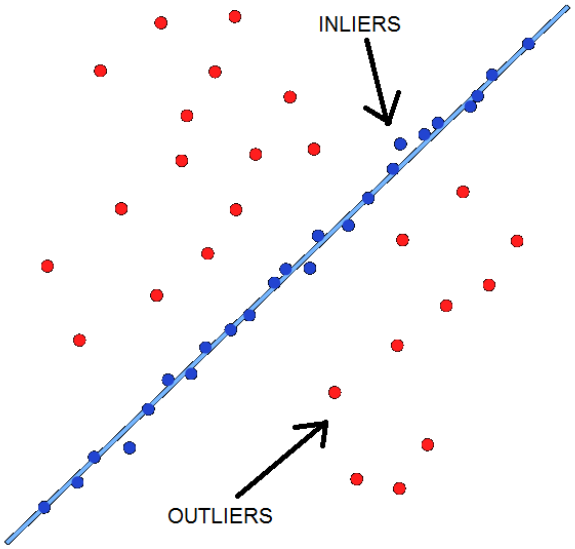

FIG. 15 LINE CREATED WITH RANSAC

The algorithm is based on the next: we select randomly 2 points of a dataset, then we create a hypothesis, better said, is drawn a line between the 2 points. After, the distance between the hypothesis and each one of the dataset points is drawn. And again we select another hypothesis and to measure the distance with the rest of the dataset points, and in this way successively. Until we find the hypothesis which has the minimum distance till the rest of the points, this hypothesis will be best to fit the model. This process can be very tedious, but to

save some time a distance threshold is established on the hypothesis (Fig. 20), because we know surely that the furthest points from the rest (outliers) will not form part of the model.

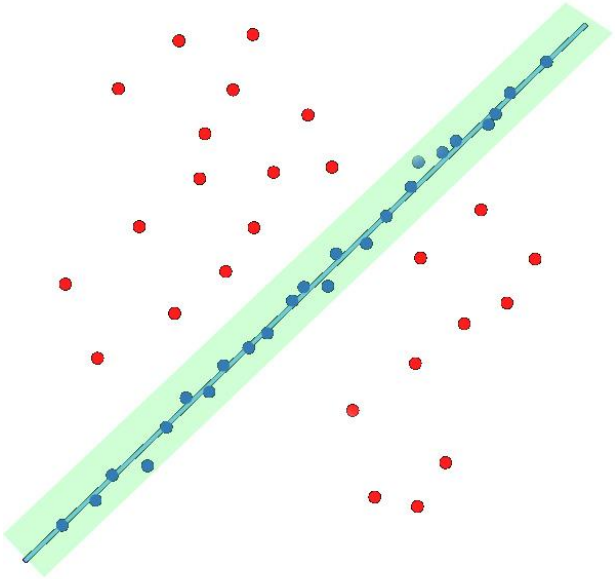

FIG. 17 RANSAC HYPOTHESIS WITH DISTANCE THRESHOLD

RANSAC in the practice

The RANSAC method allows us to refine the points list that has been obtained before, deleting all the outliers.

A kind of parameter to work the fundamental matrix is integrated in OpenCV. It is explained on the next part.

The fundamental matrix

The fundamental matrix (F) (Ma et al. 2004) is a 3x3 matrix that matches two images located at the same scene, which is limited to project the scene point in both images. Given the point projection of the scene in one of the images, the point which corresponds to the other images is matched to a line, contributing to the research of wrong correspondences

The fundamental matrix has the following properties.

For each pair of correspondent points x ↔ x' the matrix satisfies:

$$x'^T F x = 0$$

Transposed matrix: if F is the fundamental matrix of the pair of cameras (P, P'), then F^T is the fundamental matrix of the pair in the opposed order (P', P).

Epipolar lines: For any x point of the first image, the epipolar line which corresponds is $l' = Fx$. Yet at the same time, $l' = F^T x'$ represents the epipolar line that corresponds to the x in the second image.

The epipolar point: for an x point the epipolar line $l' = Fx$ contains the epipolar point e'. e' satisfies $e'^T(Fx) = (e'^T F)x = 0$ for all the x. Then, $e'^T F = 0$ and $Fe = 0$.

The fundamental matrix can be calculated without the knowledge of the inside parameters of the camera or the relative position.

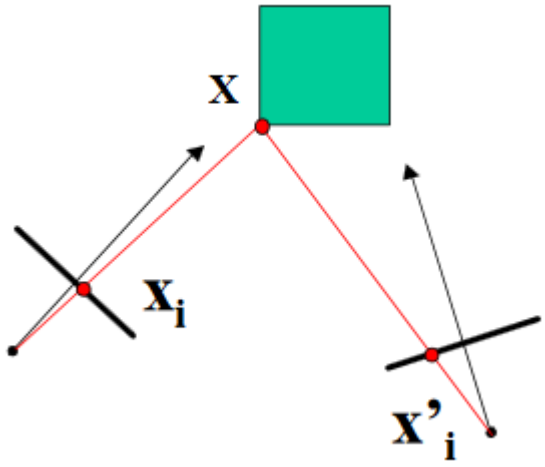

FIG. 18 TWO POINTS OF DIFFERENT IMAGES WHICH CORRESPOND TO THE SAME POINT IN THE SPACE 3D

Where Xi *and* $X'i$ are homologous points at the corresponding images. And X is the point that corresponds to the previous in the 3D space of the object.